uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería Telemática

2017-2018

Trabajo Fin de Grado

"Itinerancia en redes 4G virtualizadas"

Rodrigo Mompó Redoli

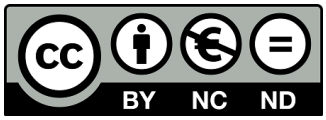
Tutor

Carlos Jesús Bernardos Cano

Director

Pablo Serrano Yañez-Mingot

5 de Marzo de 2018



EXTENDED ABSTRACT

The objective of this final year project work is to analyze the solution proposed by the 5G Exchange European project and, based on that, to make an independent implementation to validate its technological viability

Indeed, with the suppression by the European Commission on the costs associated with roaming on mobile networks within Europe, current roaming costs represent a not insignificant burden for operators. In this context, reduced roaming costs, well as increased user capacity is necessary because increased use of mobile roaming services are expected.

The 5G Exchange project (in which Universidad Carlos III participates) is working on this area by developing technology that facilitates cross-domain orchestration of services over multiple administrations. This new technology allows the deployment of network services coordinated between different operators.

In the framework of the 5G Exchange project a use case is has been developed, as a separate project, which is described in the present document.

The document analyzes the technological challenge of roaming on mobile networks, focusing on the current 4G technology. For that purpose a 4G network using a network based virtualization solution is studied. An analysis of the solution proposed by the 5G roaming Exchange project and the implementation of this solution will be performed using virtualized 4G networks. The ultimate goal is to create an environment corresponding to the case of such use, enabling a test escenario for the 5G Exchange multidomain orchestration project.

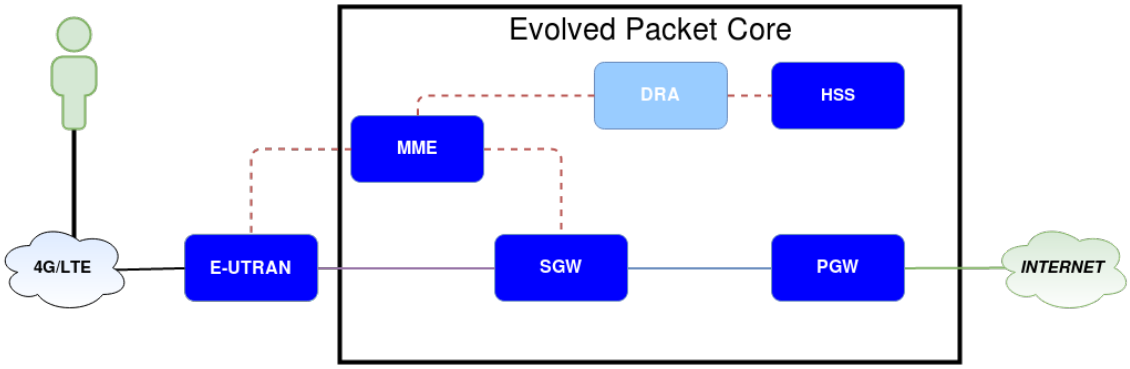


Fig. 1. Evolved Packet Core with E-UTRAN access

The Evolved Packet System (EPS) is best known as 4G, the latest mobile technology, and is the basis for this project. To better understand what the Evolved Packet System is, it will be divided into two blocks. On one hand is radio access and moreover is the core network. This division has reason to be because of the scarcity of radio spectrum, which is

limited and precious. While in the core network functionality and versatility is prioritized, access radio focus will be on the efficient use of the limited radio spectrum.

As the wireless access, EPS allows simultaneous use of different technologies. E-UTRAN, commonly called 4G or LTE, is the radio access technology defined by 3GPP standards 4G. E-UTRAN is radio access technology itself but EPS improves the user's speed and efficiency in spectrum use against past UTRAN (3G) or GERAN (2G).

These access networks are old but remain compatible with the core network known as Evolved Packet Core. It is maintaining compatibility to facilitate deployment to operators, allowing the temporary coexistence of different technologies so that the deployment can be progressive.

The Evolved Packet Core is the core network architecture evolved in EPS. In the new architecture EPS there is a separation between the control plane and the data plane. This allows better management of resources. In the data plane, traffic is routed from the client through the Service Gateway (SGW) to the Packet Gateway (PGW), where traffic turns to the corresponding network packets.

In the control plane, the Mobility Manager Entity (MME) is responsible for controlling the various elements of the network to ensure that authenticated users receive and send traffic correctly. Cellular network needs lot of signage to ensure proper operation thereof. For this, all nodes are interconnected by different interfaces defined in the standard. All this signaling is sent using the DIAMETER protocol.

On larger architectures it becomes very complex the design and configuration of signaling. In roaming scenarios as proposed in this project, an extra level of complexity is added. This is because they come into play numerous stakeholders and various operators that have to agree to set such signaling. There is a solution to the problem that will be used in this project. It consists of including a new element called Routing Agent Diameter (DRA). DRA facilitates the centralized exchange of signaling between operators.

Using a real 4G network for the project would be extremely costly, to the point of being impractical to conduct tests in a real environment. To reduce costs and make a feasible deployment, a virtual functional environment is built.

The selected software solution is OpenEPC. Despite called Open, it is a closed payment platform. OpenEPC not only creates a virtualized core network, but it also allows simulate both antennas as users. Thanks to that a functional environment can be built without the need of an expensive equipment or radio 4G licenses.

These virtualized operators have to be deployed on a virtualization platform that allows deploy all the 4G elements. In this project two different virtualization platforms are used: kernel-based virtual machine (KVM) and OpenStack.

Kernel-based virtual machine (KVM) is based on Linux. KVM is based on QEMU. The difference between them is that KVM uses processor extensions while QEMU uses emulation. KVM uses the scheduler and memory management of Linux kernel which ma-

kes the platform much simpler than other similar options. KVM requires hardware support to run, namely processors which support Hardward Assited Virtualizartion (HVM).

OpenStack is not only a virtualization software but a complete cloud computing platform. This term is used quite generically to anything that runs in the cloud. However, OpenStack meets all requirements to be considered a truly Cloud Computing service. OpenStack Cloud Computing can be classified as Infrastructure as a Service (IaaS). These are the platforms that can be rent for storage services, network and computing to deploy virtual machines.

OpenStack includes a module called Heat. Although Heat is not one of the main modules of OpenStack it is one of the key modules for the present project. It has the ability to create complex environments with virtual machines, networks, subnets and routers etc., automatically from a file called template.

This project creates, configures and provisions a solution for roaming based on KVM. In order to take advantage of OpenStack, once the work environment is set on KVM it is borne to a cluster on OpenStack. This strategy combines the simplicity of KVM with the benefits of OpenStack process orchestration.

Conceptually there are two possible roaming solutions. These two solutions differ in the packet network that is used to send traffic. 5G Exchange project proposes the use of both solutions by changing between them dynamically. Thus, at all times the most advantageous solution from the point of view of efficiency, and from the point of view of the user service is used.

Home-routed roaming is currently the roaming solution used by all operators. In the literature it is referred as Roaming Legacy. The scheme in this case is quite simple: a tunnel using any mobility technology is created between the SGW visited operator and the PGW local operator. In this case GTP provides mobility support and information to collect traffic. Nevertheless, this solution is not efficient over long distances and has scalability problems given the exponential increase of traffic per user.

Considering the above problems, the logical solution is to allow to use the itinerant PGW of the visited operator. Thus many of the problems are solved. Although this option is supported by the EPS architecture, in practice operators want to keep their own control and pricing systems. This solution is known as roaming LocalBreakout. To allow control and pricing by operators, a copy of the local PGW that virtualize the data center in the visited operator is created. Thus the operator keeps control over traffic, but the costs associated with distance and high latency problems are avoided.

In order to optimize the resources used for roaming traffic, the project proposes 5G Exchange dynamic selection between the two roaming solutions. This development comes as the use case for Multidomain Orchestration discussed earlier. Depending on the number of users and the amount of traffic among other variables, the domain orchestrator selects the most efficient solution at all times. Then, the multidomain orchestator will deploy the

corresponding solution.

First thing to do is to deploy the solution on the KVM environment in order for OpenEPC to be properly set before adding more complexity with OpenStack and the orchestration.

OpenEPC provides a series of running machines that are pre-provisioned with an operator. This OpenEPC deployment is quite basic and implements the SGW and PGW on the same node. To validate the roaming solution it is needed that both nodes are separated, allowing to test the different scenarios using tunneling (GTP in this case).

In addition DRA should be added, which provides the advantages mentioned above, as well as to change some settings on the IP address of some nodes. The following two figures show the original deployment provided by OpenEPC and the deployment to be achieved in order to further test the roaming solution.

The visited operator is called *Green Operator* while the *Red Operator* is the local one.

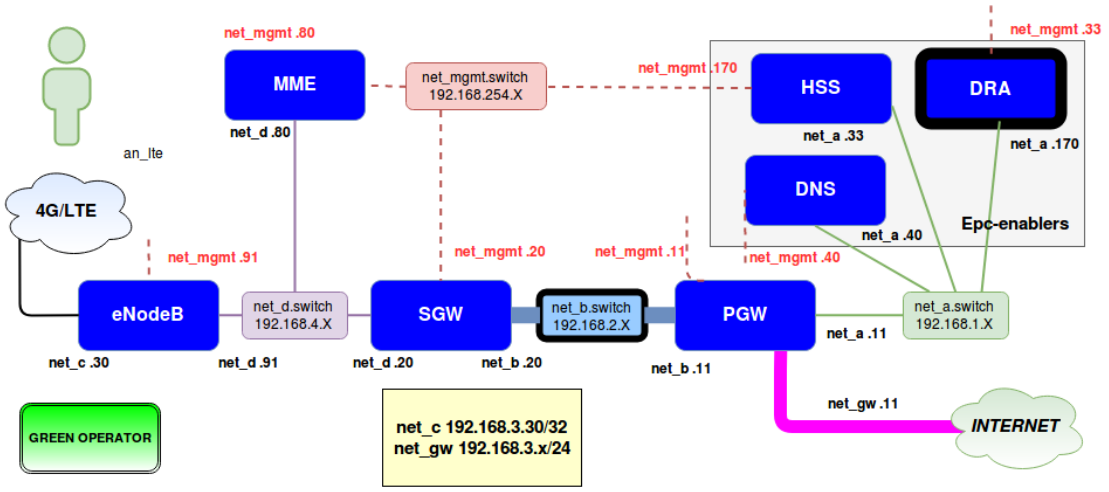


Fig. 2. Green Operator setup

Green differs mainly in the configuration and system provisioning. To demonstrate that both operators are completely independent, different subnets are used and also other parameters are modified to uniquely identify each operator. In the following figure the Red Operator deployment is shown. All the same, once both operators are working properly the roaming solution is configured as it is also shown in the figure.

Both operators need to be communicated. Signaling between them is made through the DRA / DR modules. Furthermore, in the case of home-routed roaming an interconnection network is used in the data plane.

Once the proposal on the environment tests done on KVM is validated on a single machine, the next step is to test the solution in an environment closer to a real scenario in which different servers are hosted in different computer centers. In this case, the virtualization platform used is OpenStack.

Due to the high cost of servers, the project only has two physical servers when the

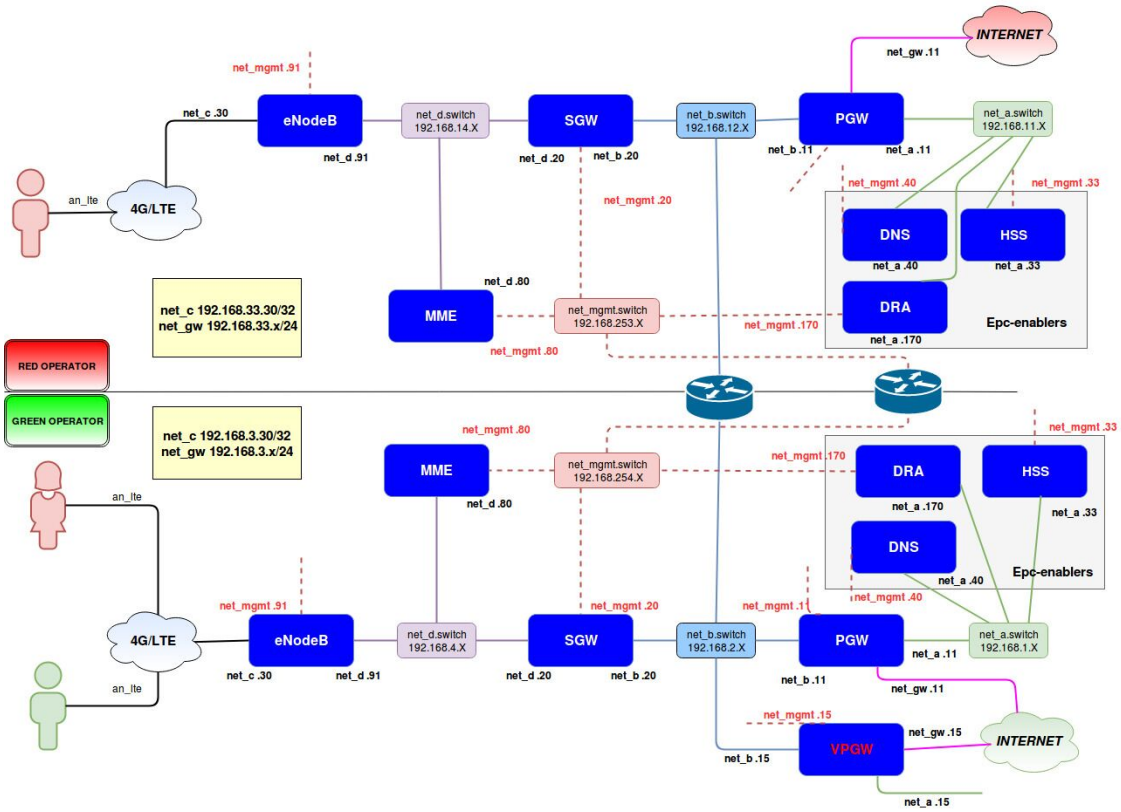


Fig. 3. Complete deploy scheme over KVM

environment requires at least four to deploy OpenStack. To solve the problem, a cluster of OpenStack is installed on each server using KVM virtual machines to obtain the necessary number of machines. Due to the large amount of resources needed by the virtualized operator, a single compute node would have an excessive size for backup. Two compute nodes are used instead to avoid the above problem.

Although deployment can be done in two clusters within the same server, for this project a more complex environment is used which is similar to a real deployment. Each of the physical servers is located on a different computer center. Both servers are interconnected by a switching network called Exchange SandBoxő 5G. The network integrates a data plane and a control plane to which the corresponding interfaces of the clusters are connected.

OpenStack has multiple tools that are useful when deploying complex environments. 5GEx project requires the complete environment to be deployed automatically. For this purpose, the Heat OpenStack module is used. In this project report the necessary changes are detailed in depth for OpenEPC running on KVM, along with the files needed to deploy the Heat solution automatically.

Conceptually, the green operator only provides its computing resources. Administration, configuration and maintenance of the V-PGW is provided by the red operator, as well as the costs associated with deploying it. For this reason a new tenant in the cluster is created that belongs to the green operator, which is totally independent of this. That way

the red operator can deploy this V-PGW isolated from the rest of the network to have a better control of the consumed resources.

The 5G Exchange multidomain orchestrator handles the V-PGW deployment as well as modify the DNS entry that selects the PGW to be used. To validate that all is working fine the V-PGW is created and the DNS is changedl manually.

Regarding the final year project, both deployment and validation were successful and all proposed objectives were achieved. For next steps it is recommended that more flexible virtualization platforms are used in order to make deployed services more scalable since some scalability limitations were found when using OpenStack.

DEDICATORIA

En agradecimiento a Luca Cominardi, Carlos Jesús Bernardos y Pablo Serrano por ofrecerme esta oportunidad formativa con la que tanto he aprendido.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

1.1. INTRODUCCIÓN

En las siguientes páginas se analizará el reto tecnológico que supone la itinerancia *(roaming)* en redes móviles, centrándose en la actual tecnología 4G. Para ello se estudiará el funcionamiento de una red 4G usando una solución basada en virtualización de red. Se propondrá una solución al nuevo reto tecnológico que presenta el *roaming* en la unión europea y se probará sobre la red 4G mencionada anteriormente. Por último, se analizará y se validará el resultado para comprobar la viabilidad de dicha solución.

1.2. MOTIVACIÓN

Tras la supresión por parte de la Comisión Europea de los costes asociados a la itinerancia en redes móviles dentro del territorio europeo, los costes actuales del *roaming* suponen una carga no despreciable para los operadores. Más adelante se analizará más en profundidad las medidas impuestas por Europa. En este contexto, es necesario una reducción de los costes de *roaming*, además de una mayor capacidad, ya que se espera un incremento del uso de los servicios móviles.

Según el Eurobarómetro realizado a los dos meses de la supresión de los recargos por *roaming* en territorio europeo, se ha detectado un incremento notable en el uso de las redes móviles en itinerancia. El 31 por ciento de los usuarios hacen uso de datos en itinerancia, frente al 15 por ciento que lo hacía antes. En llamadas internacionales europeas, el 24 por ciento de los usuarios las realiza, frente al 11 por ciento anterior a la supresión. [1]

Este cambio normativo Europeo genera la necesidad de buscar nuevas soluciones tecnológicas que puedan hacer frente al nuevo modelo de uso del *roaming* en Europa. Los servicios en *roaming* deberán prestarse en las mismas condiciones que en el territorio nacional de cada usuario. Además, se espera un aumento del uso de estos servicios tras retirar los sobrecostes. Por tanto, las redes han de prepararse para soportar un mayor tráfico de itinerancia y reducir los costes operativos para rentabilizar estos servicios.

1.3. OBJETIVOS

- Estudio de la arquitectura de una red móvil Evolved Packet System (EPS), particularizando en el caso de redes 4G/LTE.
- Estudio sobre la virtualización del Evolved Packet System, particularizando en el caso de redes 4G/LTE.

- Estudio de las diferentes plataformas de virtualización y análisis de su despeño para virtualizar redes móviles.
- Despliegue, configuración y aprovisionamiento de un operador completamente funcional, virtualizado sobre algunas de las plataformas de virtualización previamente analizadas.
- Despliegue, configuración y aprovisionamiento de un segundo operador completamente funcional virtualizado, sobre algunas de las plataformas de virtualización previamente analizadas.
- Diseño y análisis de las diferentes soluciones para interconectar ambos operadores.
- Configuración y aprovisionamiento de la solución de itinerancia propuesta por el proyecto de investigación 5GEx [2]
- Evaluación y validación de la solución de itinerancia

1.4. ESTRUCTURA DEL DOCUMENTO

El documento se divide en varias partes. El contenido de cada parte se resume a continuación:

- **Primera parte: Introducción**. En esta parte se introduce el proyecto y se comenta su motivación. También se desarrollan los objetivos, la estructura de la memoria y el marco regulatorio.
    - Capítulo 1. Introducción
- **Segunda parte: Estado del arte**. Se realizará un estudio de la estructura y funcionamiento de una red móvil EPS. También se analizarán las posibilidades de virtualización de una red EPS y algunas de las posibles plataformas de virtualización sobre las que desplegar la red virtualizada.
    - Capítulo 2. Estado del Arte
- **Tercera parte: Trabajo Realizado**. En este apartado, se desarrollará el trabajo realizado durante el proyecto. Se divide en dos capítulos. En el primero se hará un estudio y análisis de la solución a implementar, y en el segundo se desarrollará en profundidad la implementación realizada.
    - Capítulo 3. Diseño de la solución de itinerancia
    - Capítulo 4. Implementación de la solución de itinerancia
- **Cuarta parte: Conclusiones y trabajos futuros**. En esta parte se explica las conclusiones obtenidas del trabajo y futuros proyectos derivados del actual.

- Capítulo 5. Conclusiones y trabajos futuros

- **Quinta parte: Anexos**. En esta última parte se adjuntan las guías de instalación y otros documentos de interés, que se han realizado durante del desarrollo del proyecto, para facilitar el despliegue de un réplica del proyecto:

  - Anexo A: Guía de Instalación KVM
  - Anexo B: Guía de Instalación OpenWRT
  - Anexo C: Guía de Instalación OpenStack
  - Anexo D: Ficheros configuración OpenEPC
  - Anexo E: Orquestación (Heat) Templates

1.5. MARCO REGULATORIO

1.5.1. REGLAMENTO EUROPEO SOBRE LA ITINERANCIA

Después de varios años de convivencia del *roaming* en Europa, la Comisión Europea decidió regular las tarifas y los precios aplicados en este servicio[3]. Las operadoras de sus países miembros deberán ofrecer al usuario final un coste que corresponda con el de su país y operador de origen, en vez de aplicar costes adicionales como ocurría en el pasado, siempre y cuando la estancia en el extranjero no supere los cuatro meses.

Este servicio de *roaming* gratuito es efectivo ya desde el 15 de junio de 2017 en todos los países miembros de la Unión Europea: Alemania, Austria, Bélgica, Bulgaria, Chipre, República Checa, Croacia, Dinamarca, Eslovaquia, Eslovenia, España, Estonia, Finlandia, Francia, Grecia, Hungría, Irlanda, Italia, Letonia, Lituania, Luxemburgo, Malta, Países Bajos, Polonia, Portugal, el Reino Unido, Rumanía y Suecia.

Esta medida lleva asociados unos precios mayoristas que se irán reduciendo a lo largo del tiempo. Esa reducción paulatina de esos precios mayoristas a 5 años, da margen a las operadoras para implementar soluciones alternativas y rentables al *roaming* en la unión europea. Estos precios actualmente son de 7.7 euros el GB, mientras que en 2022 el precio pasará a 2.5 euros por GB de datos.

En este nuevo contexto regulativo, el proyecto anteriormente mencionado, 5G Exchange (5GEx), pretende ofrecer una alternativa a los operadores para optimizar los costes del servicio de *roaming*.

1.5.2. EL ESTÁNDAR 4G: 3GPP

La unión internacional de las telecomunicaciones (UIT) [4] es la encargada de definir las características que una red ha de tener para pertenecer a una determinada generación.

Existen una serie de características que ha de cumplir cierto estándar para poder publicitarse como 4G. Entre ellos está la velocidad de subida y bajada pero no es el único. En 2010, pese a no cumplir algunos de los requisitos, las UIT permitió al 3GPP llamar a sus nuevos estándares 4G.

El *Third Generation Partnership Project* (3GPP) [5] es uno de los organismos encargados de generar estándares móviles y quizás el más conocido e implementado. Este se compone de numerosas asociaciones de telecomunicaciones. Sus estándares están organizados en *releases*. Cada *release* supone la definición de una red móvil completa que es interoperable con *releases* anteriores. En la industria las redes suelen tener implementaciones de diferentes *realeases*, ya que los despliegues son incrementales. Los operadores suelen implementar solo las características que les interesan.

La primera *release* denominada como 4G fue la *release* 9 [6], tras ella ha habido una serie de *releases* añadiendo mejoras. Esta cuarta generación termina en la *release* 14 [7] que ya habla sobre los requerimientos de la próxima generación que empezará con la *release* 15 en 2018. Estas *releases* ya pertenecerán a la quinta generación de redes móviles 5G.

2. ESTADO DEL ARTE

Para este proyecto se usarán tanto tecnologías de red móvil, como tecnologías de virtualización. En este capítulo, se verá como es la arquitectura de una red móvil y sus posibilidades de virtualización. También se analizarán diferentes plataformas de virtualización sobre las que desplegar una red móvil virtualizada.

2.1. LA RED CELULAR 4G: EVOLVED PACKET SYSTEM (EPS)

Tras el primer despliegue de GSM en 1984 en Estocolmo, y con la popularización del servicio de telefonía móvil, cambió radicalmente la manera en la que se relacionan las personas. Casi 25 años después, las redes celulares han sufrido una gran transformación. En sus orígenes la red celular sustituía al teléfono fijo permitiendo mayor flexibilidad y estar siempre localizables, pero tras la irrupción de los teléfonos inteligentes en 2007 el mercado cambio.

El teléfono inteligente cambia la manera en la que los usuarios usan la red celular. Se ha pasado a un nuevo modelo donde el usuario siempre está conectado a la red y cada vez demanda mejor conectividad y más servicios. Esta demanda constante ha llevado a una rápida evolución de las redes móviles con una sucesión de diferentes tecnologías. En este proyecto se usará la última tecnología disponible en el mercado conocida como Evolved Packet System.

Para entender mejor en que consiste el Evolved Packet System se va a dividir en dos grandes bloques. Por una parte tendremos el acceso radio y por otra parte el IP de red. Esta división tiene razón de ser debido a la escasez del espacio radioeléctrico que es muy limitado y preciado. Mientras que en el *core* de red se priorizará la funcionalidad y la versatilidad, en el acceso radio se pondrá el foco en el uso eficiente del limitado espacio radioeléctrico.

En cuanto al acceso radio, EPS permite el uso de diferentes tecnologías simultaneas. E-UTRAN, aunque comúnmente conocida como 4G o LTE, es la tecnología de acceso radio definida por el 3GPP en los estándares del 4G. E-UTRAN es la tecnología de acceso radio propia de EPS que mejora la velocidad del usuario y la eficiencia en el uso del espectro frente a tecnologías radio pasadas como UTRAN (3G) o GERAN (2G). Estas redes de acceso aunque son antiguas, siguen siendo compatibles con el *core* de red conocido como Evolved Packet Core del que se hablará más adelante. Se mantiene la compatibilidad para facilitar el despliegue a los operadores, permitiendo la convivencia temporal de las diversas tecnologías para que el despliegue pueda ser progresivo.

Las tecnologías radio que se han nombrado hasta ahora son todas estándares del 3GPP. El Evolved Packet Core soporta además el uso de diferentes accesos radio usando están-

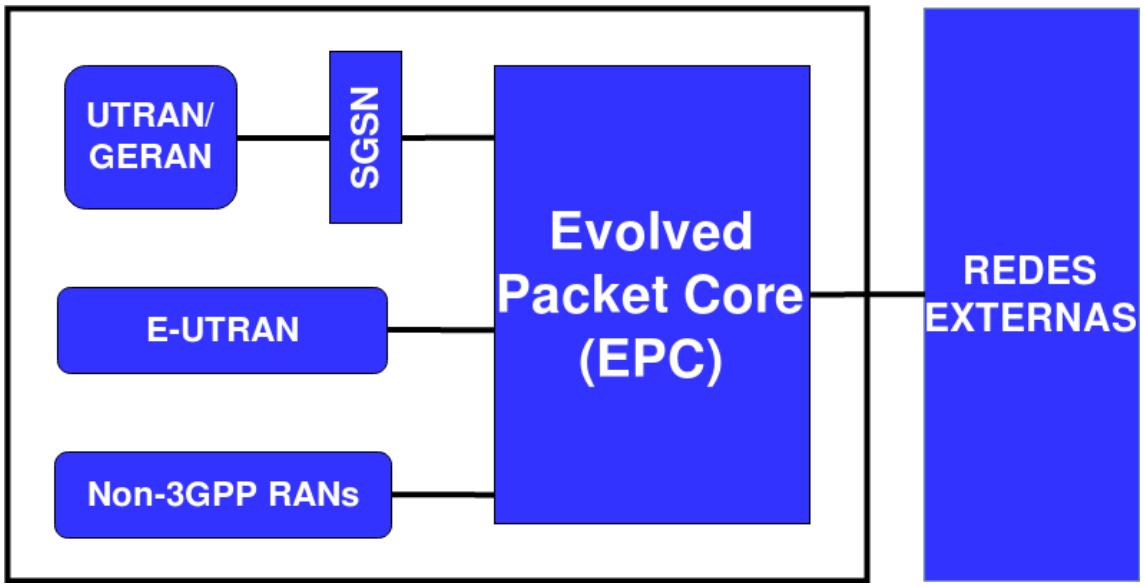


Fig. 2.1. Evolved Packet System (EPS)

dares que no sean del 3GPP. Las denominadas redes de acceso non-3GPP se dividen en dos categorías, *trusted* y *untrusted*. La diferencia fundamental entre ambas es la confianza que se tenga en la red. Si la red de acceso es gestionada por el operador u otra empresa con la que hay una relación de confianza, el 3GPP no ponen ningún requisito en cuanto a cómo se interconecta con el *core* de red. En cambio, si no se tiene ningún tipo de control o confianza sobre la red, esta será *untrusted* y requerirá una entidad especial denominada Evolved Packet Data Gateway (ePDG) como punto seguro de interconexión con el *core*.

El otro gran bloque que se ha definido antes es el *core* de red. En el caso de EPS se define el Evolved Packet Core (EPC). EPC es considerada como una red IP nativa. A diferencia que en *releases* anteriores del 3GPP, el Evolved Packet Core es una red que solo tiene conmutación de paquetes. La voz que en redes más antiguas usaba tecnología de conmutación de circuitos. En 4G las llamadas se realizan a través de la red de paquetes. La voz sobre redes de datos IP, que es el protocolo transporte más usado a nivel mundial, se denomina Voz sobre IP (VOIP). La itinerancia es algo que se resuelve en el *core* de red y por eso se le prestará especial atención y se realizará un estudio profundo sobre cómo funciona su arquitectura.

2.1.1. E-UTRAN ACCESS NETWORK: LTE

Para simplificar el estudio se va a tomar en consideración solo una tecnología de acceso radio, E-UTRAN. En E-UTRAN, la parte radio se simplifica unificándola toda en un único nodo, el eNodeB. EL eNodeB surge como la evolución del nodeB, que es la entidad radio de UMTS (3G).

Este desarrollo ha sido hecho por el grupo del 3GPP Long Term Evolution, más comúnmente conocido como LTE. Surge como una mejora de prestaciones respecto a

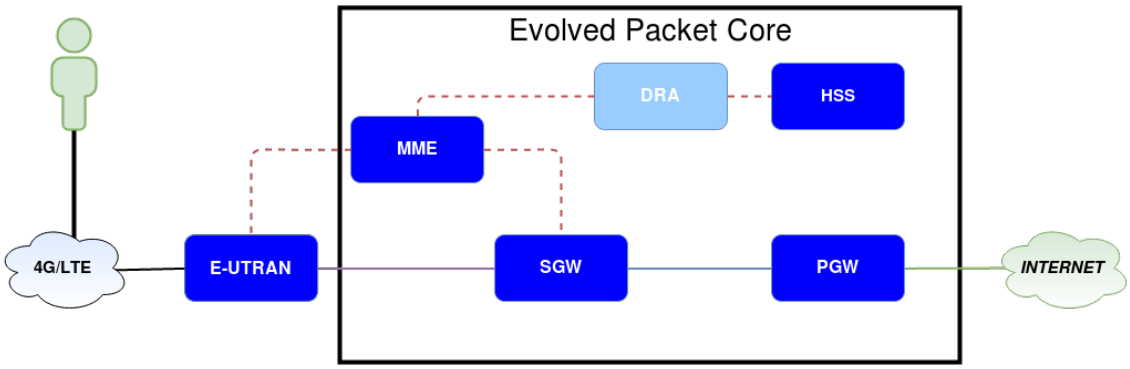


Fig. 2.2. Evolved Packet Core con acceso E-UTRAN

UMTS (3G), donde el nodo radio se llama NodeB. El grupo LTE es el encargado de mejorar la parte radio para buscar menor retardo de paquete y mayor tasa de envió de paquetes. LTE-A, la última versión hasta la fecha, es capaz de obtener velocidades teóricas de hasta 300 Mbps en bajada, frente a los 14 Mbps máximos teóricos que ofrecía UMTS.

En el plano de datos, el eNodeB se interconecta directamente con el *core* de red a través del Service Gateway (SGW) que dará servicio a diversas estaciones base y por lo tanto a diversos eNodeB. En el plano de control se conectará directamente al Mobility Manager Entity (MME), que será el encargado de gestionar toda la señalización. En EPS el SGW y el MME pueden estar alojados en la misma máquina. Si bien esto es posible, no es algo necesario.

2.1.2. EVOLVED PACKET CORE (EPC)

El Evolved Packet Core es el núcleo de red evolucionado en la arquitectura EPS. Como se ha comentado anteriormente en la arquitectura EPS existe una separación entre el plano de control y el plano de datos. Esto permite una mejor gestión de los recursos.

En cuanto al plano de datos, el tráfico se enruta a través del Service Gateway (SGW) hasta el Packet Gateway (PGW), donde se vuelca el tráfico hacia la red de paquetes correspondiente.

En el plano de control, el Mobility Manager Entity (MME) se encarga de controlar los diferentes elementos de la red para garantizar que los usuarios autenticados reciben y envían tráfico correctamente. A continuación se hablará más en detalle de los diferentes elementos del Evolved Packet Core.

SGW

Service Gateway (SGW) es el punto de interconexión entre el 3GPP radio access network (RAN) y el *core* EPC. El SGW da servicio a todas las tecnologías radio definidas en el 3GGP GERAN, UMTS y LTE. Además el SGW es el punto de anclaje para gestionar la movilidad.

La movilidad en EPC se gestiona mediante el protocolo Proxy Mobility IP Protocol (PMIP). Sin embargo, otras tecnologías están admitidas. Tanto en despliegues reales como en el despliegue que se va a realizar, se usará Gprs Tunneling Protocol (GTP). Este protocolo de movilidad, basado en tunelación al igual que PMIP, era propio de la arquitectura GPRS (2G). Pese a no ser eficiente por el *overhead* añadió debido a que GTP gestiona la tarificación, en despliegues reales, se ha mantenido para evitar tener que cambiar todo el sistema de tarificación. Este *overhead*, que solo se añade en el de tramo fibra y no en el tramo radio, con el volumen de tráfico actual no supone un problema.

MME

Mobility Manager Entity es el nodo principal del plano de control. Cada usuario conectado a la red está asociado a un MME específico que puede cambiar a lo largo del tiempo. El MME como ya se ha visto es la entidad encargada de gestionar la señalización en el plano de control. Entre otras funciones, gestiona la movilidad de usuarios cuando este se cambia de estación base y por tanto de enodeB. Esta entidad de red también es la encargada de autenticar y autorizar al usuario así como de recolectar la información de cobro de los usuarios.

El MME también es el encargado de negociar la calidad de servicio y seleccionar la red de paquetes a la que el usuario accede, usando el Access Point Name (APN). Esto es de vital importancia a la hora de gestionar la itinerancia, ya que es el MME el que decide cual será el PGW por el que saldrá el tráfico a la red de paquetes. El APN tiene una estructura muy concreta. Se compone principalmente de dos partes:

- **APN Network Identifier (APN-NI):** esta es la parte del APN que identifica a la red externa a la que el usuario se quiere conectar. Dos operadores pueden tener el mismo APN-NI. Es obligatorio y generalmente la única parte del APN que especifica el usuario.
- **APN Operator Identifier (APN-OI):** es el identificador del operador al que pertenece el PGW correspondiente a la red de paquetes a la que el usuario se quiere conectar. Esta parte del APN es opcional y generalmente no se especifica. Habitualmente el APN-OI será el correspondiente al operador con el que el usuario tiene contratado el servicio. En el caso de itinerancia, eso permite al operador visitado diferenciar que usuarios se conectan a su PGW y que usuarios al PGW del operador local con el que el usurario itinerante tiene contratado el servicio.

El MME realiza una serie de operaciones sobre el APN especificado por el usuario, antes de resolver la dirección IP del PGW asociado a la red de paquetes correspondiente. Esta IP generalmente se almacena en un registro DNS. En la figura 2.3 se puede ver las reglas con las que el MME genera el APN completo, para hacer la resolución DNS a partir de la información que proporciona el usuario. Esta información puede ser solo el APN-NI o un APN completo.

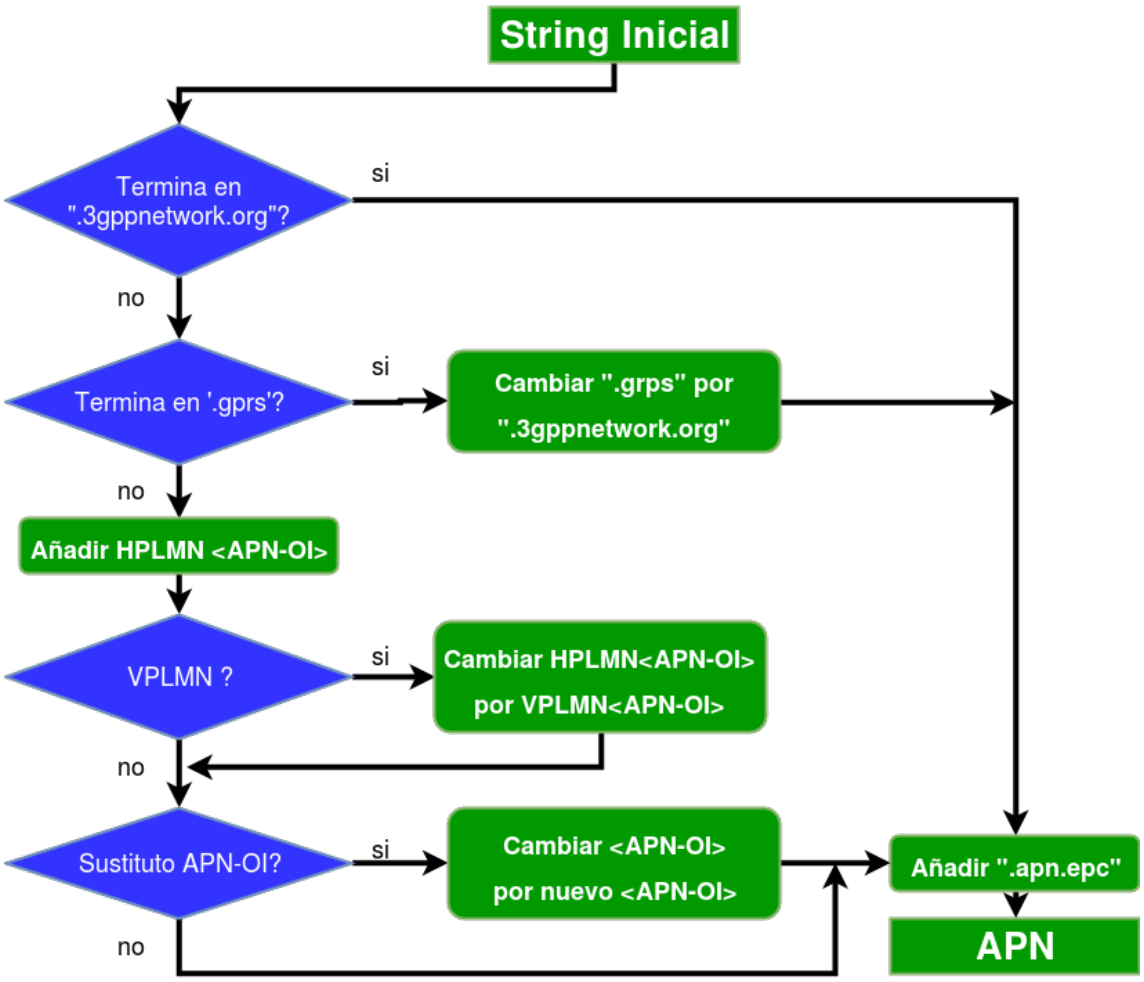


Fig. 2.3. Flowchart generación del APN en el MME

HSS

El MME, como ya se ha comentado, se encarga de autenticar y autorizar a los diferentes usuarios entre otras funcionalidades. Si en la red solo se tuviera un MME, la información de los usuarios podría estar guardada en el mismo. Como ya se ha comentado, una red tiene múltiples MMEs en función de la extensión geográfica y el número de usuarios de la red. Mantener y sincronizar varias bases de datos no es algo trivial. Por ese motivo en EPC al igual que en tecnologías móviles anteriores, se usa una base de datos de usuarios centralizada.

El Home Suscriber Server (HSS) es una base de datos con la información de todos los usuarios de la red. Solo existe una por operador, aunque generalmente esta replicada. Se replica tanto para hacer balanceo de carga como para evitar tener un único punto de fallo y aumentar la fiabilidad de la red.

Para evitar sobrecargar el sistema, el MME cuenta con una base de datos temporal, de tal manera que solo tiene que consultar al HSS una vez por conexión. El MME tendrá la información de todos los usuarios conectados a su porción de la red, al igual que llevará un control con diferentes grados de precisión, de donde se encuentra cada usuario. El HSS

por tanto solo contendrá la información de usuario estática relacionada con que tiene contratado y autorizado cada usuario.

PGW

El Packet Gateway (PGW) como ya se ha comentado, es el punto de acceso a una determinada red de paquetes, que puede ser Internet o una red privada. Desde el punto de vista del usuario, el PGW será su primer salto. A nivel IP es equivalente a que el usuario estuviera directamente conectado a este nodo. El PGW es el encargado de asignar direcciones IP a los diferentes usuarios. Además, también se encarga de desencapsular el tráfico GTP y encaminarlo a la red de paquetes correspondiente.

Aunque el APN identifica una red de paquetes que generalmente tiene asociado un PGW, este soporta más de una red de paquetes cada una con su APN correspondiente. Podría darse el caso en el que varios APNs identifiquen al mismo PGW y el tráfico asociado a cada uno de ellos se distribuyera a redes de paquetes diferentes.

DRA

Una red celular necesita gran cantidad de señalización para asegurar el correcto funcionamiento de la misma. Todos los nodos que anteriormente se han descrito están interconectados por diferentes interfaces definidas por el 3GPP. Toda esta señalización se envía usando el protocolo DIAMETER.

DIAMETER es una evolución del protocolo RADIUS, ambos diseñados para el intercambio de señalización. Este protocolo funciona a nivel de aplicación y es muy flexible. Proporciona muchísima versatilidad para amoldarse a todo tipo de despliegues. El protocolo es bastante complejo de configurar.

Este protocolo define el concepto de *peer*. Cada elemento que use este protocolo tendrá una serie de *peers* asignados que usen este protocolo. Por cuestiones de seguridad DIAMETER requiere que todos los nodos se conozcan entre sí, es decir, que se tengan asignados como *peers* para intercambiar información.

En arquitecturas más grandes se vuele muy complejo el diseño y configuración de la señalización. En escenarios de itinerancia como el propuesto en este proyecto, se aumenta aún más el nivel de complejidad. Esto es debido a que entran en juego numerosos *stakeholders* y varios operadores han de ponerse de acuerdo para configurar entre ellos la señalización.

Existe una solución para los problemas anteriores que se usará en este proyecto. Consiste en incluir un nuevo elemento denominado Diameter Routing Agent (DRA). El DRA es un *router* que redirige el tráfico de señalización. De esta manera, los nodos solo tienen que conocer un elemento, el DRA, al que enviarle la señalización. El funcionamiento es muy similar al de un *router* IP. Se configuran reglas en el DRA, para saber cómo redirigir

el tráfico. Al igual que una red IP, se puede tener una topología más compleja con varios DRA, no es necesario que el tráfico de señalización llegue en un único salto.

El DRA también permite realizar balanceo de carga o tener nodos replicados y redirigir el tráfico en caso de que un HSS se caiga por ejemplo. Un DRA que interactúa con redes externas se denomina Diameter Edge Routing (DRE). Entre otras funcionalidades, permite esconder la topología de la red de redes externas, y es especialmente útil cuando hay que gestionar la señalización en escenarios de itinerancia como el que se han visto.

Otras Entidades

Existen más elementos para los que no se va a entrar en detalle. Muchos de ellos tiene que ver con los sistemas de tarificación y cobro, el uso de otros acceso radio y funcionalidades como los SMS.

El *core* EPC también cuenta con su propio servidor DNS. A parte de resolver las IP de los diferentes elementos, también tendrá almacenada la información relativa al APN. El MME tras obtener el APN completo como se vio antes, realizará una consulta DNS usando el APN, y obtendrá la IP del PGW correspondiente.

2.2. EPS VIRTUALIZADO USANDO OPENEPC

El uso de una red 4G real para el proyecto es extremadamente costo, hasta el punto de ser inviable llevar a cabo pruebas en un entorno real. Para reducir los costes y hacer viable desplegar un entorno funcional, se puede optar por virtualizar la red. En la actualidad ya existen proyectos para desplegar redes 4G virtuales, como el proyecto que tiene Telefónica junto con Huawei para desplegar una solución virtualizada en 13 países de Latinoamérica y Europa usando CloudEPC [8].

Fig. 2.4. OpenEPC

En el proyecto se ha optado por otra solución software denominada OpenEPC [9]. Pese a llamarse Open, es una plataforma cerrada de pago. OpenEPC no solo permite crear un *core* de red virtualizado, sino que además permite simular tanto las estaciones base como lo usuarios. Gracias a eso se puede tener un entorno funcional sin la necesidad de tener equipamiento caro o licencias radio 4G. Lo único que se necesita es uno o varios equipos de propósito general.

OpenEPC está preparado tanto para funcionar con equipamiento radio simulado como con equipamiento real. Para reducir los costes en este proyecto se usará acceso radio y usuarios simulados conectados al *core* EPC virtualizado.

Aunque en este proyecto se usará un operador simplificado, OpenEPC ofrece muchas más funcionalidades. OpenEPC tiene también soporte para tecnologías radio *legacy* como son las propias de 2G y 3G. Además permite el uso de tecnologías de acceso ra-

dio *non-3GPP* tanto *trusted* como *untrusted*. En cuanto a servicios OpenEPC cuenta con integración de VOZIP y VOZLTE, junto con soporte IMS entre otros.

2.3. PLATAFORMAS DE VIRTUALIZACIÓN

Como ya se ha visto OpenEPC proporciona software para ejecutar funciones de red sobre máquinas virtuales. Por tanto, se necesita un entorno para desplegar la solución. En las siguientes páginas se analizarán dos soluciones, KVM y OpenStack.

2.3.1. KVM

Kernel-based Virtual Machine (KVM) es una solución basada en linux que permite crear un entorno de virtualización completo. KVM está basado en QEMU. La diferencia entre ambas reside en que KVM usa extensiones del procesador mientras QEMU usa emulación. KVM al usar el planificador y la gestión de memoria del *kernel* de linux hace que la plataforma sea mucho más simple que otras opciones similares. KVM necesita de soporte hardware para funcionar, en concreto procesadores que soporten Hardward Assited Virtualizartion (HVM).

En cuanto al apartado de red, KVM te permite crear redes virtuales. Existen tres tipos de redes que se pueden configurar en KVM:

- **NAT:** la red por defecto en KVM pertenece a este grupo. Estas redes proveen de salida a internet a las máquinas. Usan Network Address Translation (NAT) para mapear una determinada IP de la subred a una IP y puerto del host.
- **Aisladas:** son segmentos de red independientes completamente aislados tanto del exterior como de otras subredes. Estos segmentos se usan para interconectar máquinas virtuales dentro de un mismo host.
- **Routed:** Similares a las redes aisladas pero con soporte para encaminar tráfico entre diferentes segmentos de red. Estas redes también están aisladas al exterior y no permiten acceso a internet, solo a otras subredes dentro del mismo host.

En el proyecto, generalmente se usarán redes aisladas, menos para aquellas redes que requieran un acceso a internet, en cuyo caso se usará NAT nativo de linux. Aunque KVM es muy versátil, en el apartado de enrutamiento no se tiene la misma flexibilidad que puede proporcionar un *router*. Por ese motivo, se usará una distribución linux especial llamada OpenWRT que permite crear máquinas virtuales linux con todas las funcionalidades que se esperan de un *router*. En los anexos se encuentra una guía de instalación de OpenWRT.

KVM se usa y se gestiona usando la terminal de comandos. Aunque en la mayoría de las ocasiones es cómodo de usar, se tiene también la opción de usar un entorno gráfico.

Virtual Manager es una interfaz de usuario que permite crear y gestionar tanto máquinas virtuales como redes sobre KVM.

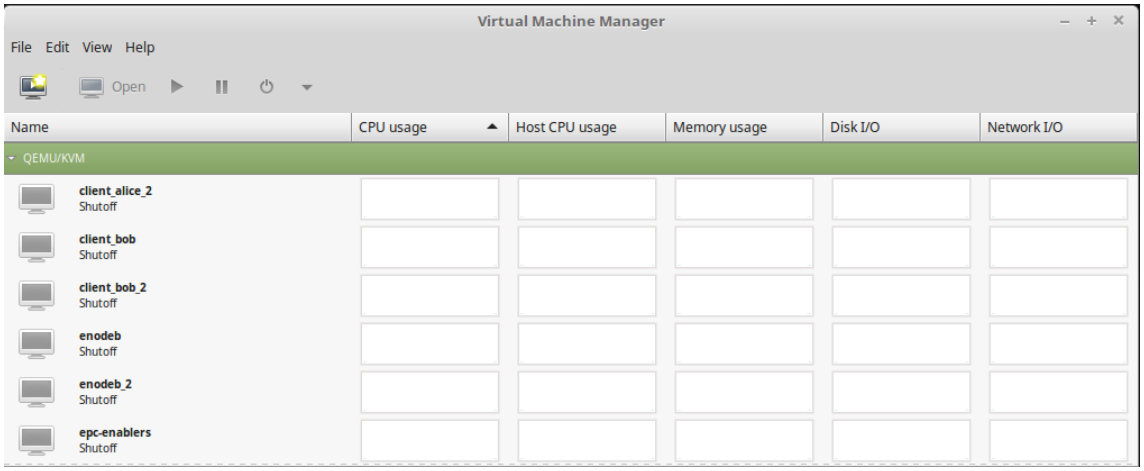


Fig. 2.5. Virtual Machine Manager

2.3.2. OPENSTACK

OpenStack no es solo software de virtualización, sino una plataforma completa de Cloud Computing. Este término se usa de manera bastante genérica para cualquier cosa que se ejecuta en la nube. Por ello se va a analizar las claves para que un servicio pueda ser etiquetado correctamente como Cloud Computing.

En primer lugar el servicio en cuestión ha de estar disponible siempre bajo demanda, sin necesidad de intervención por parte de terceros para desplegar en un momento determinado un servicio. Los servicios deberán ser accesibles a través de Internet independientemente de su localización geográfica. Además, el usuario que use Cloud Computing solo deberá abonar el coste de los recursos que usa, pudiendo aumentarlos o disminuirlos de manera flexible.

OpenStack cumple todos estos requisitos por lo que verdaderamente es un servicio de Cloud Computing. Existe también una clasificación de los diferentes tipos de servicios de Cloud Computing donde OpenStack puede clasificarse como Infrastructure as a Service (IaaS). Estas son las plataformas que permiten alquilar servicios de almacenamiento, red y cómputo para desplegar máquinas virtuales para usos muy variados.

Existen muchas plataformas y este proyecto se centrará en OpenStack que es una plataforma de software libre distribuida bajo la licencia apache [10]. OpenStack permite crear nubes tanto públicas como privadas o incluso híbridas. En este proyecto se usarán nubes privadas desplegadas en servidores que se encuentran en la universidad.

Dado que OpenStack es un entorno más complejo que KVM, ya que está pensado para funcionar en centros de cómputo y ser escalable, se va a analizar más en detalle su arquitectura y como se desarrollaría un despliegue básico.

Aunque OpenStack puede ser desplegado en solo un servidor desde la comunidad de desarrolladores se recomienda usar mínimo dos servidores. Como se puede observar en la figura 2.6, la arquitectura mínima consiste en un nodo de control y al menos un nodo de cómputo. Si se requieren más recursos, se pueden instalar nuevos nodos de cómputo para aumentar la capacidad del despliegue.

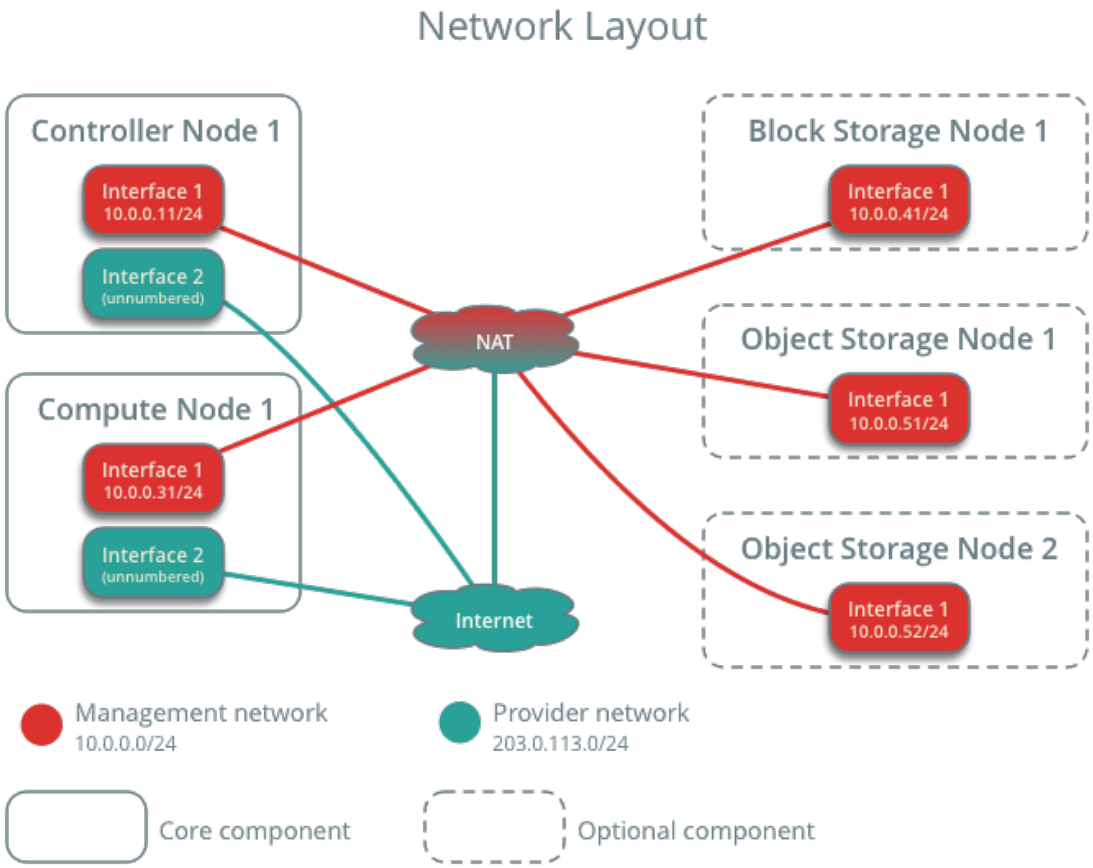


Fig. 2.6. Despliegue OpenStack Ocata Fuente: openstack.org

OpenStack se organiza en diversos componentes o módulos. Cada uno de ellos se encarga de gestionar un recurso específico, conectividad, cómputo, memoria, autenticación, etc. Se listarán a continuación los módulos más relevantes de OpenStack para este proyecto:

- **Nova (Cómputo):** Es el módulo principal de OpenStack. Se encarga de gestionar los recursos de procesador y RAM de las diferentes máquinas virtuales. Por debajo usa un hypervisor para funcionar. En la mayoría de los casos usará KVM para interactuar con el host. En el despliegue se usará KVM como hypervisor.
- **Neutron (Redes):** En OpenStack los recursos de red se gestionan en un módulo independiente del usado para los recursos de cómputo. Dicho módulo se llama Neutrón y es uno de los módulos más complejos. Entre sus funcionalidades básicas encontramos la creación de redes, subredes y *routers*. Además ofrece soporte NAT

para salir a Internet y un paquete de medidas de seguridad como *port security* o grupos de seguridad para gestionar reglas del *firewall*.

- **Heat (Orquestación):** Heat no es uno de los módulos principales de OpenStack pero para este proyecto resultará muy útil. Tiene la capacidad de crear entornos complejos con máquinas virtuales, redes, subredes y *routers* entre otros, de manera automática a partir de un fichero denominado *template*.

OpenStack, aunque es muy potente, está diseñado para albergar servidores *end-to-end*. El diseño de la arquitectura no contempla que máquinas virtuales dentro del *cluster* reenvíen tráfico. Las máquinas virtuales de OpenEPC si reenvían tráfico. Esto será un punto a tener en cuenta a la hora de desplegar posteriormente la solución de itinerancia sobre OpenEPC.

Dada la complejidad que entraña el despliegue de un *cluster* de OpenStack y para facilitar la instalación, en los anexos se encontrará una guiá de instalación de OpenStack.

2.3.3. COMPARATIVA KVM VS OPENSTACK

Como se ha visto previamente, ambas plataformas son muy distintas. OpenStack es una plataforma muy usada en centros de cómputo y trae más herramientas como la Orquestación. A priori proporciona un entorno más robusto, y permite de manera sencilla escalar el despliegue. Si bien OpenStack tiene muchas ventajas, añade una capa de complejidad a tener en cuenta. Realizar un despliegue de OpenStack no es trivial. Además, Neutrón no está pensado para desplegar máquinas virtuales que enruten tráfico, y esto puede suponer una complicación.

Por otra parte, KVM es un entrono mucho más simple, ya que solo proporciona las herramientas mínimas de virtualización. En el aspecto de red es mucho más flexible, ya que usa las herramientas propias de linux para gestionar el gran número de redes virtuales que se necesitarán.

Por estos motivos, en este proyecto se procederá a crear, configurar y aprovisionar la solución de itinerancia en KVM. Una vez que el entorno sobre KVM funcione, para aprovechar las ventajas de OpenStack, se procederá a portar la solución de itinerancia a un *cluster* de OpenStack. De esta manera se tiene un proceso más simple de desarrollo y a la vez se puede contar con los beneficios de la orquestación que anteriormente se han comentado.

3. DISEÑO DE LA SOLUCIÓN DE ITINERANCIA

A la hora de diseñar un esquema de itinerancia no existe una única solución buena. El operador ofrece al usuario una oferta de servicios con características muy distintas. Por ejemplo, no es lo mismo un servicio de conexión de datos que de voz. Además desde el punto de vista del operador no es lo mismo tener un usuario en itinerancia, que varios miles de ellos.

Conceptualmente existen dos soluciones de itinerancia. Estas dos soluciones se diferencian en la red de paquetes por la que sale el tráfico. Como se verá más adelante, el proyecto 5G Exchange propone el uso de ambas soluciones cambiando entre una y otra de manera dinámica. De esta manera, en cada momento se usará la solución que más convenga tanto desde el punto de vista de la eficiencia, como desde el punto de vista del servicio al usuario.

Como se ha visto anteriormente, el APN identifica la red de paquetes a la que el usuario quiere acceder para usar diversos servicios. No obstante, un usuario no tiene por qué tener un APN único. En función de la red de paquetes a la que quiera acceder ese APN puede cambiar. Cada servicio puede tener un APN debido a las características del servicio. Cada servicio puede requerir cursar el tráfico por una red de conmutación distinta.

En el caso concreto de este proyecto no se va escoger una solución u otra por el tipo de servicio usando APN distintos. En este caso de uso, como ya se ha comentado anteriormente, se busca una solución eficiente para ofrecer un servicio concreto a un usuario que este en itinerancia. Ese servicio ha de ser de iguales características para el usuario en ambos operadores. Un ejemplo de ello es el acceso a internet. La mayoría de operadores ofrecen un servicio similar. El objetivo es conseguir que de manera transparente al usuario, el operador seleccione eficientemente a través de que PGW cursa el tráfico y de esta manera conseguir un uso eficiente de los recursos para poder soportar el incremento de la demanda de tráfico en itinerancia.

3.1. HOME ROUTED ROAMING

Actualmente esta es la solución de itinerancia desplegada que usan todos los operadores. En la literatura también se puede encontrar como *roaming legacy*. El esquema en este caso es bastante simple.

Como se ha visto anteriormente, se crea un túnel GTP entre el SGW y el PGW, que entre otras funcionalidades, proporciona soporte de movilidad e información para el cobro del servicio. Normalmente, ambas entidades son del mismo operador y se usa una red de transporte propia del operador para encaminar el tráfico entre un elemento y el otro. En el caso de que se tenga un usuario en itinerancia, simplemente se modificarán los

endpoints del túnel GTP. En ese caso el túnel quedaría establecido entre el SGW del operador visitado y PGW del operador local. De esta manera el tráfico saldrá por su sitio habitual. Además el sistema de cobros y el de calidad de servicio será el habitual.

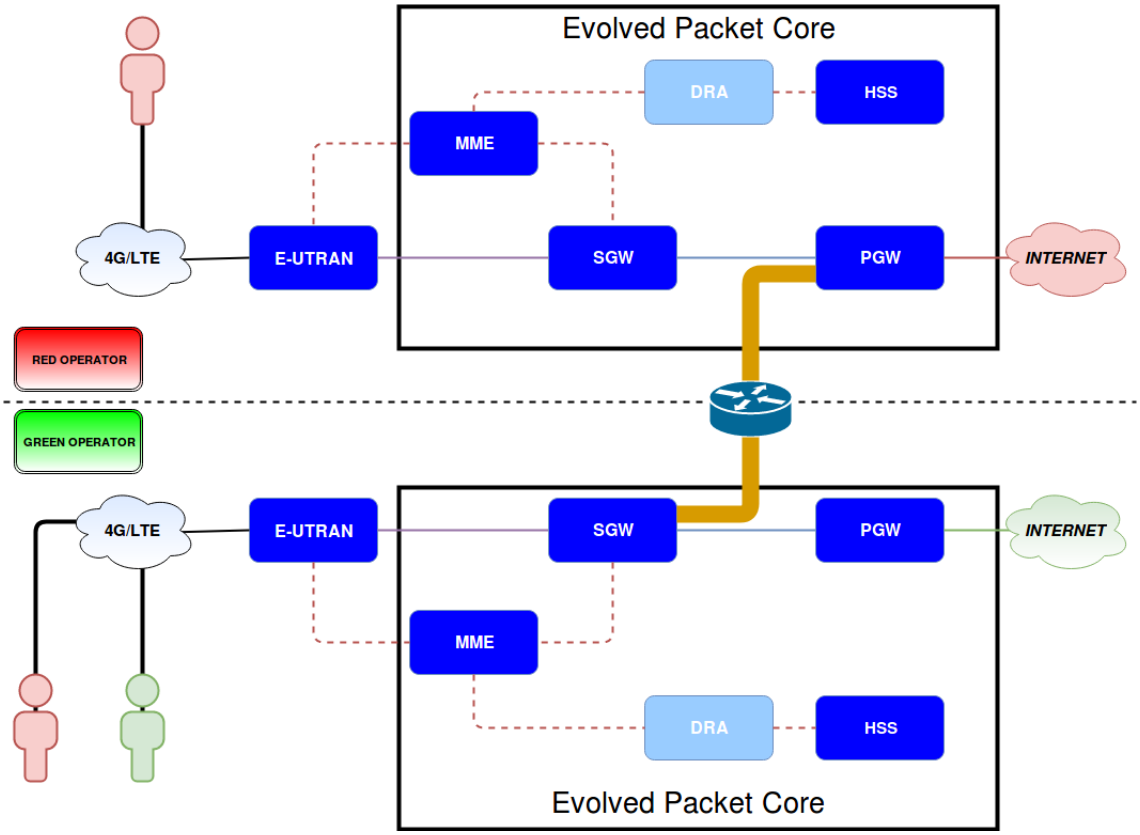


Fig. 3.1. Solución itinerancia Home Routed

Este esquema plantea varios problemas. En cuanto a los retos tecnológicos, existe un problema en cuanto a la gestión de señalización. Este tema se tratará más adelante en profundidad puesto que es un problema común a ambas soluciones de itinerancia. Además en escenarios transcontinentales pese al uso de fibra óptica existe un aumento de la latencia debido a la distancia recorrida. Aunque para la mayoría de las aplicaciones esto no supone un problema, para algunas de ellas es un tema crítico.

En cuanto a los retos comerciales, aunque mayoritariamente se cuenta con empresas especializadas en transporte de datos internacionales, generalmente cruzar varios países supone un problema. Entre esos problemas, se encuentra la mayor dificultad para generar acuerdos de itinerancia cuando intervienen múltiples partes y los problemas en cuanto a normativa y legislación según la normativa de cada país.

3.2. LOCALBREAKOUT ROAMING

Teniendo en cuenta los problemas anteriores, la solución lógica es permitir al usuario itinerante que use el PGW del operador visitado. De esta manera se solucionarían muchos

de los problemas. Aunque esta opción la arquitectura EPS la soporta, en la práctica los operadores quieren tener su sistema de control y tarificación centralizados.

Una posible solución para evitar la cesión del control, seria instalar un PGW del operador local en la red del operador visitado. Este esquema, aunque soluciona prácticamente todos los problemas, tiene un coste elevado. En la mayoría de los casos el uso de servicios en itinerancia no es constante y depende de factores como la época del año.

Lo ideal sería tener ese PGW alquilado de tal manera, que solo se use cuando se necesite, mientras que cuando no haya demanda por parte de los usuarios, no tenga coste. Gracias a la virtualización, que ya se ha analizado en profundidad, esto es posible. Se puede crear una máquina virtual que haga las funciones de PGW, que se denominará de aquí en adelante como Virtual PGW (V-PGW).

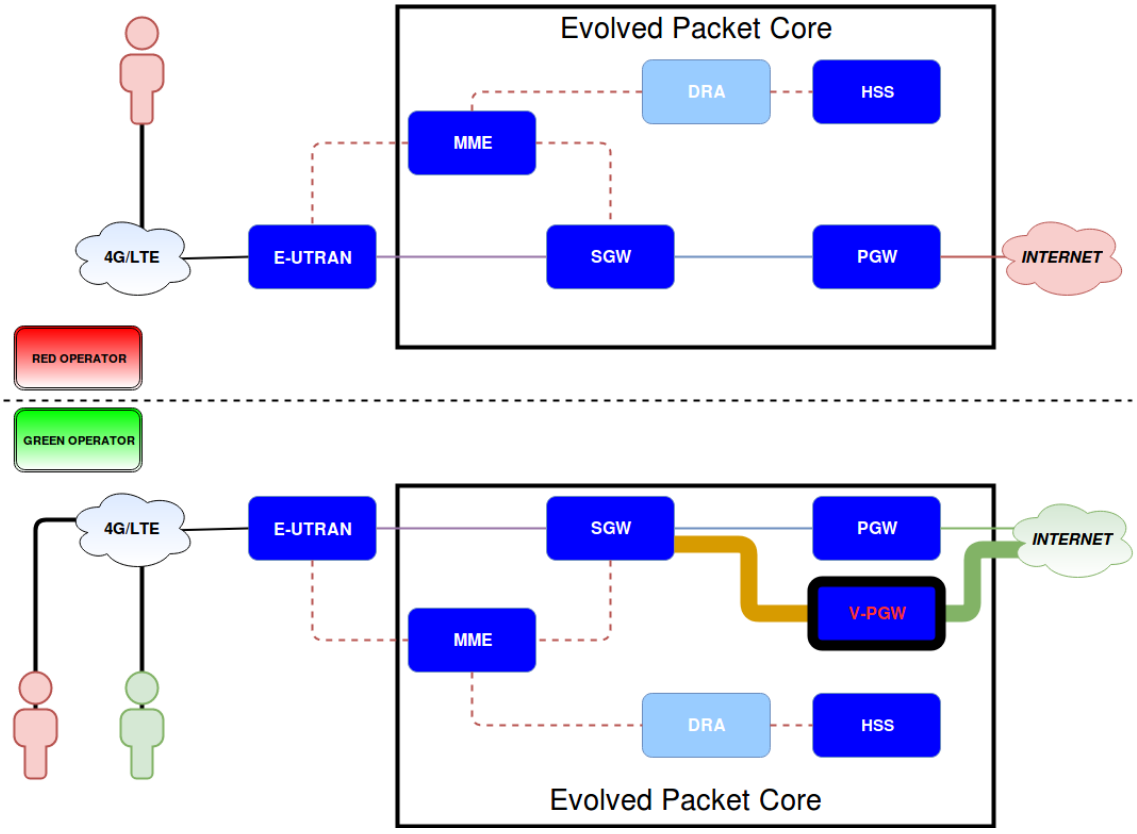


Fig. 3.2. Solución itinerancia LocalBreakout

Esta máquina aunque gestionada por el operador visitado, tendrá el software y las configuraciones correspondientes del operador local, dándole control sobre el usuario. Este elemento se instanciaría en el centro de datos correspondiente del operador visitado. De esta manera, cuando la carga de usuarios sea alta se pueda usar este V-PGW, mientras que si el tráfico es bajo, se usará la solución anterior si es más eficiente.

3.3. ITINERANCIA DINÁMICA: ORQUESTACIÓN MULTIDOMINIO

Se han analizado conceptualmente ambas soluciones de itinerancia desde un punto de vista meramente funcional. Como se ha comentado antes, no existe una única solución buena para la itinerancia. Cada solución tiene una serie ventajas e inconvenientes además de unos costes. Posteriormente en el análisis socio-económico se desarrollará más en profundidad el rendimiento particular de ambas soluciones.

Con el fin de optimizar al máximo los recursos usados para el tráfico de *roaming*, el proyecto 5G Exchange propone la selección dinámica entre ambas soluciones de itinerancia. Este desarrollo se presenta como un caso de uso para la Orquestación Multidominio.

La Orquestación Multidominio es similar a la orquestación que previamente se analizó como parte de los módulos de OpenStack. En este caso, la diferencia fundamental es la existencia de un orquestador único, que interactué con diversos centros de cómputo y *tenants*. Esto permite, de manera centralizada desplegar recursos de red por todo un territorio en función de las necesidades globales que tenga esa área, sin importar que los recursos físicos pertenezcan a distintos proveedores.

En el caso de uso desarrollado en este proyecto, este orquestador central se encargaría de desplegar en V-PGW en el centro de cómputo del operador visitado en un *tenant* separado. De esta manera, los recursos usados por el V-PGW son totalmente independientes del operador visitado. Este tipo de despliegue permitirá que el operador local no necesite tener centros de cómputo en el operador visitado, si no que podrá alquilar esos recursos, de la misma manera que se pueden alquilar máquinas virtuales en los diversos proveedores en la nube. En la figura 3.3 se puede ver la imagen completa del escenario propuesto por el proyecto 5G Exchange.

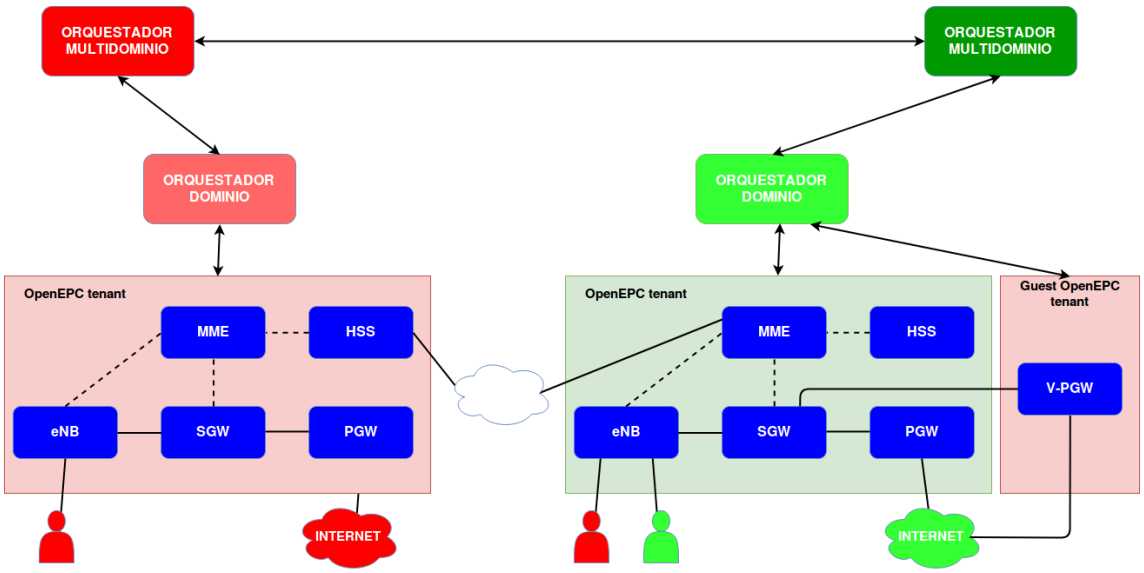


Fig. 3.3. Escenario propuesto por el proyecto 5G Exchange

Este orquestador central no se limita únicamente a asignar recursos de disco, cómputo

y red. Puesto que el objetivo último es el despliegue de servicios coordinados, en la mayoría de los casos el orquestador también tendrá que configurar adecuadamente los servicios desplegados.

En el caso de uso que se está manejando el 5G Exchange usando OpenEPC, es necesario que el MME encargado de resolver el APN devuelva en cada momento la IP correspondiente. En el entorno actual, la resolución del APN se realiza mediante el uso de DNS. En este caso, el orquestador multidominio tendrá que actualizar los DNS cuando realice el cambio entre ambas soluciones.

El trabajo realizado en este proyecto incluye el despliegue de ambas soluciones y proporcionar una interfaz que permita al orquestador desarrollado por otros proyectos de investigación interactuar directamente con el despliegue. Se detallará más adelante en el capítulo de implementación.

3.4. ANÁLISIS SOCIO-ECONÓMICO

Antiguamente la comunicación a grandes distancias suponía un reto. La gestión política y económica de los territorios lejanos siempre ha sido compleja. En la época colonial española existió la figura del virrey a quien se delegaba el poder en un territorio. Era la persona encargada de tomar decisiones que no podían esperar entre 2 y 3 meses, que era el tiempo necesario para que un mensaje fuera enviado y contestado entre continentes.

Las comunicaciones modernas permiten en cuestión de milisegundos el intercambio de información con cualquier punto del planeta. La sociedad actual no sé entendería sin esa comunicación instantánea que proporcionan las redes de comunicaciones actuales. En los últimos 10 años se ha dado un paso más. Ya no solo es clave interconectar dos puntos fijos en tiempo real, ahora se trata de interconectar personas.

Las redes móviles actuales permiten a las personas estar siempre conectadas y poderse comunicar en cualquier lugar en cualquier momento. Estas nuevas tecnologías suponen una revolución tanto social como comercial. Las comunicaciones ya no dependen del lugar.

Actualmente esa movilidad está garantizada dentro de los diversos territorios nacionales. Moverse entre Madrid y Barcelona no supone ningún problema para el usuario. Bien porque el operador tiene las cobertura o en el caso de las operadoras móviles virtuales por que la cesión de red es nacional. Cuándo esa misma movilidad se plantea de manera internacional el escenario es muy distinto. Existen sobrecostes de itinerancia bien conocidos por parte de los usuarios y suponen una limitación en la movilidad de las personas. Además, aplicaciones sensibles a la latencia, como pueden ser videojuegos, control en tiempo real, telemedicina, etc, no son viables con las tecnologías actuales de *roaming*.

La Unión Europea siguiendo las políticas de movilidad europeas ha puesto fin a esos sobrecargos de *roaming*. El impacto social y económico de la medida es enorme. Supone

un beneficio para los usuarios y para la sociedad, a la vez qué una pérdida de ingresos para las compara las telecos.

La solución propuesta en este proyecto propone alternativas a las soluciones tecnológicas actuales de *roaming* mejorando tanto los costes como el servicio ofrecido a los usuarios. Con el trabajo realizado en este proyecto se acerca un poco más a la realidad de una Europa sin barreras tecnológicas. Además ayudara a los operadores a mitigar la pérdida de ingresos por *roaming* reduciendo los costes.

Las soluciones actuales de *roaming* siempre redirigen el tráfico hacia el país de origen. Estás políticas de tráfico no siempre son las más eficientes en muchos casos. Existe un sobrecoste tanto económico como de recursos energéticos, al alimentar toda la infraestructura necesaria para transportar ese tráfico a largas distancias. Este proyecto colabora con el proyecto 5G Exchange englobado en el 5G-PPP. La implementación técnica de la solución propuesta por el 5GEx será parte de los diferentes prototipos tecnológicos desarrollados para la siguiente generación de redes móviles 5G.

Debido a la utilidad del proyecto para los objetivos globales tanto del proyecto 5G Exchange como del 5G-PPP, el presupuesto de este trabajo es superior al habitual. El coste total para el desarrollo de este trabajo asciende a los 40.750 euros. El desglose completo del presupuesto se encuentra en la siguiente tabla.

TABLA 3.1. PRESUPUESTO

Nombre	Coste (euros)	Cantidad	Total (euros)
Licencia OpenEPC	25.000	1	25.000
Servidor (8 meses)	1.060	2	2.120
Ordenador portátil (8 meses)	470	1	470
Horas trabajador Junior	9,375	960	9.000
Horas trabajador Senior	20,80	200	4.160
Total (euros)			40.750

4. IMPLEMENTACIÓN DE LA SOLUCIÓN DE ITINERANCIA

En este capítulo se desarrollará en profundidad los elementos clave del despliegue realizado. Debido a la complejidad del entorno, el despliegue se ha realizado de manera incremental. En la documentación se desarrollarán todos los pasos seguidos hasta alcanzar el despliegue final, que cumpla con los requerimientos necesarios para el proyecto 5G Exchange.

Antes de comenzar con el primer despliegue propuesto, se ha de realizar una instalación de KVM. Se recomienda tener instalado Virt Manager que proporciona una interfaz gráfica, aunque no es indispensable. En caso de estar usando un servidor sin interfaz gráfica, la instalación de Virt Manager se podrá realizar en el equipo monitor. A través de este equipo con conectividad al servidor, se puede hacer uso de Virt Manager en remoto. En los anexos se pueden encontrar una guía de instalación con información más detallada.

4.1. DESPLIEGUE DE NODO OPENEPC EN KVM

Como se vio anteriormente la red de OpenEPC está compuesta por nodos que cumplen distintas funciones. Dentro de un nodo puede estar alojada una sola función o diversas funcionalidades como es el caso de los *epc-enablers*; se hablará de ellos más adelante. En primer lugar se va a explicar cómo se instala y configura un nodo para que cumpla una determinada funcionalidad.

Para desplegar un nodo de OpenEPC primero se deben desplegar los segmentos de red que necesita dicho nodo. OpenEPC incluye unos ficheros xml con la descripción de los diferentes segmentos de red. Con estos ficheros se puede desplegar distintas redes virtualizadas para tener la topología de red necesaria.

Todos los nodos llevan una interfaz conectada al segmento de red conocido como *mgmt* o red de *managment*. Como se ha visto antes, existen diversos tipos de segmentos de red. En este caso, la red *mgmt* es de tipo NAT mientras que el resto de redes son aisladas. Para desplegar los diferentes segmentos, se ejecutarán los siguientes comandos:

```
1 #virsh net-define segmento.xml
2 #virsh net-start nombre_segmento
3 #virsh net-autostart nombre_segmento
```

Nótese que si se quiere que tras un reinicio de la máquina, automáticamente se activen los segmentos de red, se tendrá que ejecutar el tercer comando. Esto es opcional.

Cada nodo de OpenEPC consiste en una máquina virtual con una distribución de linux de Centos7. Estas máquinas de OpenEPC incluyen todos los ficheros binarios y la mayoría de los ficheros de configuración para convertirlo en cualquiera de los nodos de la red.

Para desplegar un nuevo nodo, se clonará una máquina con todos los ficheros de OpenEPC, descargados del repositorio. Una vez que se tenga dicha máquina preparada, se puede clonar tantas veces como sea necesario hasta obtener todos los nodos necesarios para el despliegue. Si se quiere clonar la máquina más cómodamente , se usará la interfaz gráfica de Virt Manager. En la figura 4.1 se puede ver como clonar la máquina con Virt Manager.

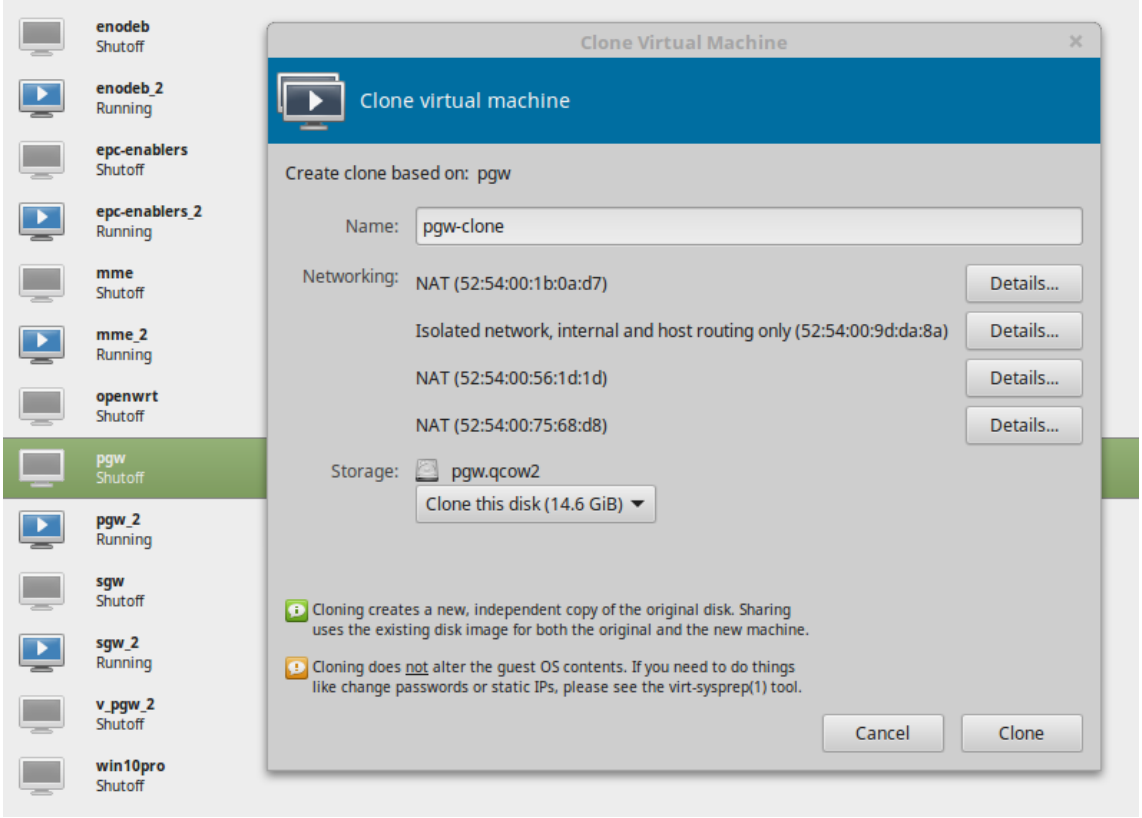


Fig. 4.1. Clonación nodo usando Virt Manager

En caso de que no se tenga interfaz gráfica también se puede usar un terminal con el siguiente comando:

```
#virt-clone \
--original máquina_base \
--name nuevo_nodo \
--file ruta/nuevo_nodo.qcow2
```

Dependiendo de la funcionalidad que se le quiera dar a la máquina, esta tendrá que tener una serie de interfaces conectadas a los segmentos de red adecuados. Una vez clonada la máquina se debe añadir todas las interfaces que necesite el nodo. Una vez más, esto se puede realizar tanto desde la interfaz gráfica como desde la consola de comandos. Una vez conectada la máquina a la red correctamente, se ha de comprobar su correcto funcionamiento y proceder a configurar el nodo.

Si todo funciona correctamente, el primer paso será renombrar las interfaces de red

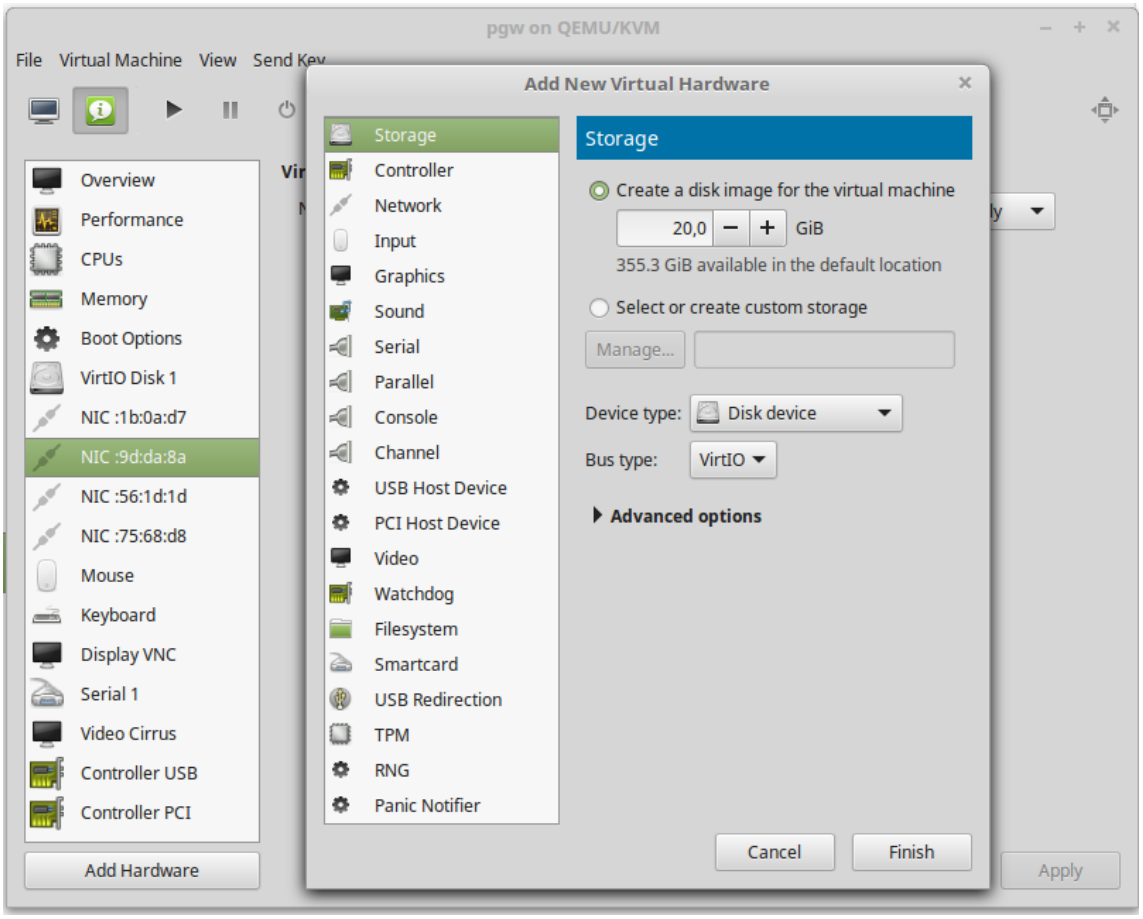


Fig. 4.2. Creación de una nueva interfaz usando Virt Manager

que han de corresponderse con los nombres asignados por OpenEPC en vez de los que asigna linux por defecto. Para ello se puede usar un *script* que proporciona OpenEPC.

```
/opt/OpenEPC/install/make_udev.sh
reboot
```

OpenEPC está diseñado para adaptarse de manera rápida a cualquier topología. En la carpeta *install* se encuentra un fichero denominado *data_topology_vars.sh*. En este fichero se encuentran definidos todos los parámetros y las IPs de los diferentes segmentos de red. Por defecto, no hay que modificar este fichero, pero más adelante cuando se despliegue el segundo operador se tendrá que modificar gran parte de este fichero.

Como se ha comentado anteriormente, cada nodo de OpenEPC contiene uno o varios servicios. Cada uno de estos servicios se configura con un fichero xml. Todos los ficheros xml se encuentran en el mismo directorio. Para editar la configuración de un determinado servicio, se accederá a su correspondiente fichero xml:

```
vim /opt/OpenEPC/etc/nombre_del_servico.xml
```

Algunos servicios hacen uso de bases de datos. Existen ficheros en el directorio *ins-*

tall de OpenEPC con diferentes configuraciones de aprovisionamiento. Para evitar posteriormente tener que configurar el operador entero manualmente, se pueden editar estos ficheros con la configuración que se necesite, dejando todo automáticamente configurado tras la instalación.

Por último, se realiza la instalación dándole al nodo genérico una tarea concreta. La instalación se realizará ejecutando un script que proporciona OpenEPC siguiendo las instrucciones que aparecen en la terminal.

```
#. /opt/OpenEPC/install/install_system.sh
```

Es recomendable realizar una copia del nodo tras comprobar su correcto funcionamiento. En caso de querer probar una nueva configuración o ante un fallo de la máquina, se puede restaurar la última versión evitando volver a realizar toda la instalación y configuración.

4.2. DESPLIEGUE OPERADOR VERDE EN KVM

OpenEPC proporciona una serie de máquinas preaprovisionadas con un operador funcionando. El despliegue que proporciona OpenEPC es básico e implementa el SGW y el PGW en el mismo nodo. Para validar la solución de itinerancia, se necesita que ambos nodos estén separados permitiendo probar los diferentes escenarios usando tunelación GTP. Además se tendrá que añadir el DRA, que proporciona las ventajas anteriormente comentadas, y cambiar alguna configuración relativa al direccionamiento IP de algunos nodos. En las figuras 4.3 y 4.4, se puede ver el despliegue original proporcionado por OpenEPC y el despliegue que se quiere alcanzar para poder probar posteriormente la solución de itinerancia.

Dado que configurar el operador desde cero no es trivial, solo se cambiarán aquellas configuraciones estrictamente necesarias para el operador verde, realizando un despliegue lo más parecido al proporcionado por OpenEPC. En la próxima sección, se desplegará el segundo operador denominado como operador rojo. Al ser un operador independiente, si se tendrá que modificar la mayor parte de la configuración. Con el operador rojo se detallará en profundidad como desplegar un operador usando configuraciones personalizadas.

Se puede observar que cada nodo de OpenEPC contiene una sola funcionalidad sobre las que se habló en el capítulo 2. Los *epc-enablers* sin embargo son una excepción. En despliegues donde existe un gran número de usuarios, y por tanto el tráfico es elevado, se tiende a disponer de una o varias máquinas para cada función de red. En despliegues más pequeños esto no es necesario. OpenEPC permite agrupar varias funcionalidades de red en un mismo nodo. El nodo denominado *epc-enablers* es una máquina donde se alojara diversas funciones de red que dan soporte al sistema. Entre esas funcionalidades, las más destacadas son el DNS, el HSS y el DRA. Estas dos funciones han sido desarrolladas en profundidad anteriormente. En este nodo se pueden añadir los elementos necesarios para dar soporte SMS o VOZIP entro otros. Si se quisiera escalar el sistema, se podría usar un

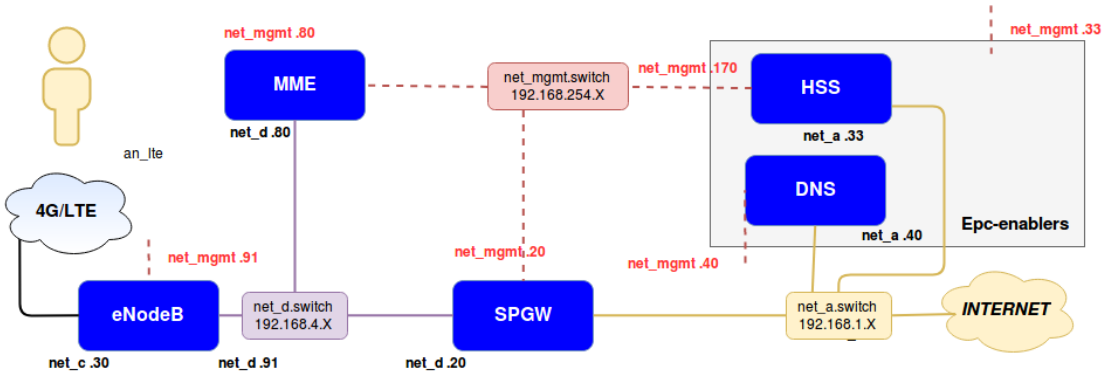


Fig. 4.3. Despliegue Original OpenEPC

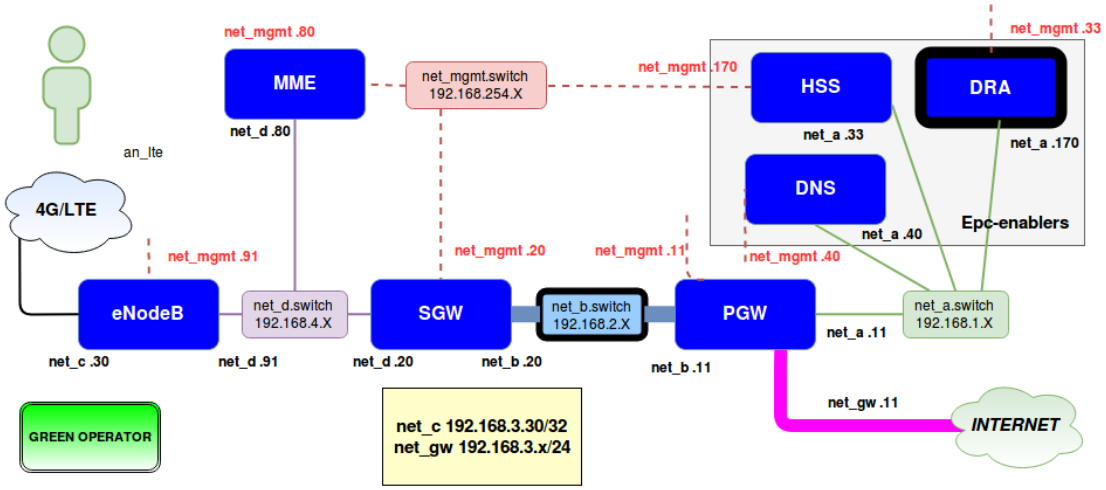


Fig. 4.4. Despliegue Operador Verde

nodo para cada una de las funcionalidades agrupadas en los *epc-enablers*.

Para desplegar el operador verde, primero se han de crear los distintos segmentos de red usando los ficheros de OpenEPC, como se vio anteriormente. Al separar el PGW y SGW se necesita interconexión entre ambos nodos para crear el túnel GTP. Para ello se debe añadir un nuevo segmento de red, net_b. El fichero xml correspondiente con la definición de la net_b que no proporciona OpenEPC se detalla a continuación:

```
<network>
<name>net_b</name>
<uuid>uuid</uuid>
<bridge name='virbr9' stp='on' delay='0'/>
<mac address='52:54:00:11:14:5b'/>
<ip address='192.168.2.1' netmask='255.255.255.0'>
</ip>
</network>
```

Como se ha visto anteriormente, un PGW puede dar servicio a diferentes redes de conmutación de paquetes. En el entorno que incluye OpenEPC se usa el segmento de red net_a como red externa de conmutación de paquetes. El fin es demostrar control sobre la

salida del tráfico y permitir posteriormente usar distintas redes de paquetes para distintos servicios.

Se creará una salida alternativa a internet añadiendo un nuevo segmento de red que se denominará net_gw de tipo NAT. Esta subred proporcionará salida a Internet a los distintos clientes. En despliegues reales la salida a Internet de los clientes se realiza usando *nating*. Por ese motivo, en este escenario se usará la misma aproximación. El PGW dará IPs privadas pertenecientes a la subred net_gw para dar acceso a Internet a los clientes mediante *nating*.

```
<network>
        <name>net_gw</name>
        <uuid>uuid</uuid>
        <forward mode='nat'/>
        <bridge name='virbr0' stp='on' delay='0'/>
        <mac address='52:54:00:3a:00:db'/>
        <domain name='net_gw'/>
        <ip address='192.168.3.1' netmask='255.255.255.0'>
        </ip>
</network>
```

Una vez desplegados todos los segmentos necesarios se puede proceder a desplegar los diferentes nodos. Se pueden reutilizar los nodos que proporciona OpenEPC para el *client_bob*, eNodeB, MME y *epc-enablers*. Adicionalmente, se necesitará clonar dos máquinas genéricas, una para desplegar el PGW y otra para desplegar el SGW. En los anexos se encuentran los diferentes ficheros xml para cada uno de los nodos. Se recomienda usarlos puesto que contiene toda la configuración compleja. Esta puede no coincidir con los ficheros que incluya OpenEPC.

Se han de reinstalar todos los nodos tras configurar correctamente cada nodo. En los nodos client_bob, eNodeB, MME y SGW, no se necesita ningún cambio adicional más allá de posibles modificaciones en la topología de red que se verán más adelante y algún cambio en el protocolo Diameter. Tanto en el PGW como en los *epc-enables* se tendrá que realizar algunas modificaciones extras que se detallarán a continuación.

4.2.1. PGW

Anteriormente se habló del uso por algunos nodos de bases de datos. El PGW contiene una base de datos donde se guarda información de aprovisionamiento del mismo. El PGW puede ser aprovisionado en la instalación usando un *script* que proporciona OpenEPC. Aunque esto es opcional es recomendable tener dicho fichero para evitar tener que aprovisionar manualmente todo el sistema. En este operador se ha usado la configuración por defecto guarda en el fichero /opt/OpenEPC/install/data_provisioning_pdn_ops_pgw.sh.

Si ese fichero no existiera se recomienda crearlo con la configuración de aprovisionamiento correspondiente. En caso de partir del entorno que proporciona OpenEPC, que

incluye el nodo SPGW, se puede coger como referencia el contenido del fichero data_provisioning_pdn_ops_spgw.sh. Para el entorno propuesto, el contenido de ambos ficheros será el mismo, solo hay que renombrar dicho fichero para que el instalador de OpenEPC lo encuentre.

El PGW tiene además otro cambio significativo, la inclusión de una nueva red de conmutación de paquetes. Toda la configuración relativa a esta nueva red de conmutación de paquetes se encuentra en el fichero pgw.xml que se puede encontrar en los anexos. No obstante se van a comentar los cambios realizados en este fichero y en el entorno para que la nueva red funcione correctamente.

Se va a configurar el PGW lo más parecido a un despliegue real posible. En los despliegues de red actuales se está repartiendo direccionamiento IP privado dinámico a los usuarios. El operador realiza posteriormente *nating* masivo para dar salida a todo ese tráfico. Esta técnica se usa para mitigar el problema de la escasez de direcciones IPv4 que existe actualmente. En el despliegue realizado se usará también esta técnica de *nating*. Para ello se debe permitir al PGW gestionar tráfico ARP, ya que todas las IPs de usuario se encuentran en la misma subred. Para configurar esto se debe modificar el módulo *routing_pgw* en el fichero pgw.xml

```
<Module binaryFile="modules/routing_pgw/routing_pgw.so">
   <![CDATA[
     <WharfROUTING arp_enabled="yes" hash_size="32" buffer_size
        ="6000000">
             <PathManagement
              hash_table_size="16"
              expires_interval="90"
              t3_response="3"
              n3_requests="3"
              check_interval="1"
             />
   </WharfROUTING>
   ]]>
</Module>
```

A partir del a versión 7 de OpenEPC la configuración de las interfaces para las diferentes redes de paquetes, ya no se realiza desde el fichero de configuración XML. Para configurarlas se usará el de OpenEPC, que modificará dicha información en la base de datos del PGW.

OpenEPC para facilitar su adaptación a diferentes topologías de red tiene todas sus variables almacenadas en un mismo fichero. Es te fichero llamado data_topology_vars.sh se encuentra en el directorio /opt/OpenEPC/install. El fichero incluye direccionamiento IP, identificadores, FQDNs, APNs, MCC, MNC, entre otras variables.

Puesto que se ha cambiado el direccionamiento IP, se debe modificar las variables de entorno correspondientes contenidas en el fichero data_topology_vars.sh. Por coherencia

se debe modificar este fichero e incluir la nueva versión en todas las máquinas. Esto es especialmente importante para algunas variables. Para las variables que se han de añadir o modificar para el PGW no es necesario pero si recomendable.

```
export pgw_net_gw_ipv4="192.168.3.11"
export pgw_net_gw_prefixv4="$pgw_net_gw_ipv4/24"
export pgw_net_gw_ipv6="fc00:1234:3::11"
export pgw_net_gw_prefixv6="$pgw_net_gw_ipv6/112"
export pgw_net_gw_gateway="192.168.3.1"
export intfgw="net_gw"


expodashboardrt pgw_pdn_ipv4_net="192.168.3.0/24"
export pgw_pdn_ipv6_net="fc00:1234:3::/48"
```

Por último se deberán configurar las rutas para la nueva interfaz adecuadamente. OpenEPC usa un *script* para gestionar las configuraciones de red sustituyendo el Network Manager en linux. Este *script* se encuentra en la ruta /opt/OpenEPC/etc/network/set_ip.sh. A continuación se adjunta la configuración completa del PGW en este fichero.

```
"pgw")
   case "$intf" in
      "mgmt")
        # PGW -> Gx, S6x
        ip addr $ip_op $pgw_mgmt_prefixv4 dev $intf
       ;;
      "net_a")
        # PGW -> SGi
       ip addr $ip_op $pgw_net_a_prefixv4 dev $intf
       ip addr $ip_op $pgw_net_a_prefixv6 dev $intf

      ;;
      "net_gw")
        # Default Internet route, for EPC clients actually
        # PGW -> SGi
        ip addr $ip_op $pgw_net_gw_prefixv4 dev $intf
        ip addr $ip_op $pgw_net_gw_prefixv6 dev $intf

        ip route $ip_op default via $pgw_net_gw_gateway dev $intf
      ;;
      "net_b")
       # PGW -> S5/S8, S2a, S2b
       ip addr $ip_op $pgw_net_b_prefixv4 dev $intf
       ip addr $ip_op $pgw_net_b_prefixv6 dev $intf
      ;;
      *)
        echo "Not implemented for node $node and $intf"
        exit -1
esac
```

```
30    ;;
```

Como se puede ver se configura como ruta por defecto la net_gw. OpenEPC es un software en constante desarrollo. En el momento del desarrollo del proyecto el módulo routing_eth, no funcionaba correctamente. En sustitución, OpenEPC usa un módulo llamado routing_raw. El módulo en cuestión no realiza ningún tipo de *routing*, simplemente coge el tráfico desencapsulado y lo reenviá a la tabla de rutas de linux. Por ese motivo, se debe configurar la tabla de rutas de linux para reenviar el tráfico.

4.2.2. EPC-ENABLERS

Como se comentó anteriormente los *epc-enablers* es un nodo que engloba diversos servicios. En el entorno propuesto se usan tres servicios principales, DNS, HSS y DRA. Al realizar las modificaciones correspondientes en el PGW, el aprovisionamiento por defecto difiere del original. Al final de esta sección se explicará comó modificar el aprovisionamiento correspondiente. Primero, se ha de reinstalar el nodo *epc-enablers* para habilitar todas las funcionalidades necesarias.

Por defecto, la versión que incluye la máquina proporcionada por OpenEPC, incluye solo los componentes mínimos para funcionar, HSS y DNS. Como se ha visto anteriormente, para desplegar la solución de *roaming* y gestionar la señalización de *diameter*, se va a usar el Diameter Router Agent (DRA). Para habilitarlo, aparte de configurar el fichero XML adecuadamente, se tiene que reinstalar el nodo *epc-enablers* en su versión denominada *epc-enablers*, que incluye todos los servicios disponibles.

El protocolo DIAMETER como se analizó anteriormente permite el intercambio de señalización entre los diversos servicios. En el entorno actual este intercambio sucede entre el MME y el HSS para intercambiar la información de usuarios. Ambos nodos se tiene como pareja. Al introducir el Diameter Routing Agent (DRA), estos nodos deberán tener como pareja únicamente al DRA. El DRA en cambio ha de tener como pareja tanto al MME como al HSS. En los anexos se encuentran los ficheros de xml con la configuración completa.

El DRA necesita una serie de reglas para enrutar el tráfico de señalización. Estas no vienen configuradas por defecto, y tendrán que ser configuradas. DIAMETER es un protocolo muy complejo y flexible con múltiples opciones de configuración de reglas. Se van a incluir las usadas en este despliegue. Las reglas relativas al escenario de *roaming* se verán más adelante.

Como se puede observar en la imagen, se usará una ruta genérica. Esta envía todo el tráfico por defecto al HSS. Posteriormente se añadirá una ruta con más prioridad para enviar el tráfico al operador rojo. En el aprovisionamiento original esta ruta se incluye con más de un HSS para balanceo de carga. Se deberán eliminar de la lista todos los HSS que no se estén usando.

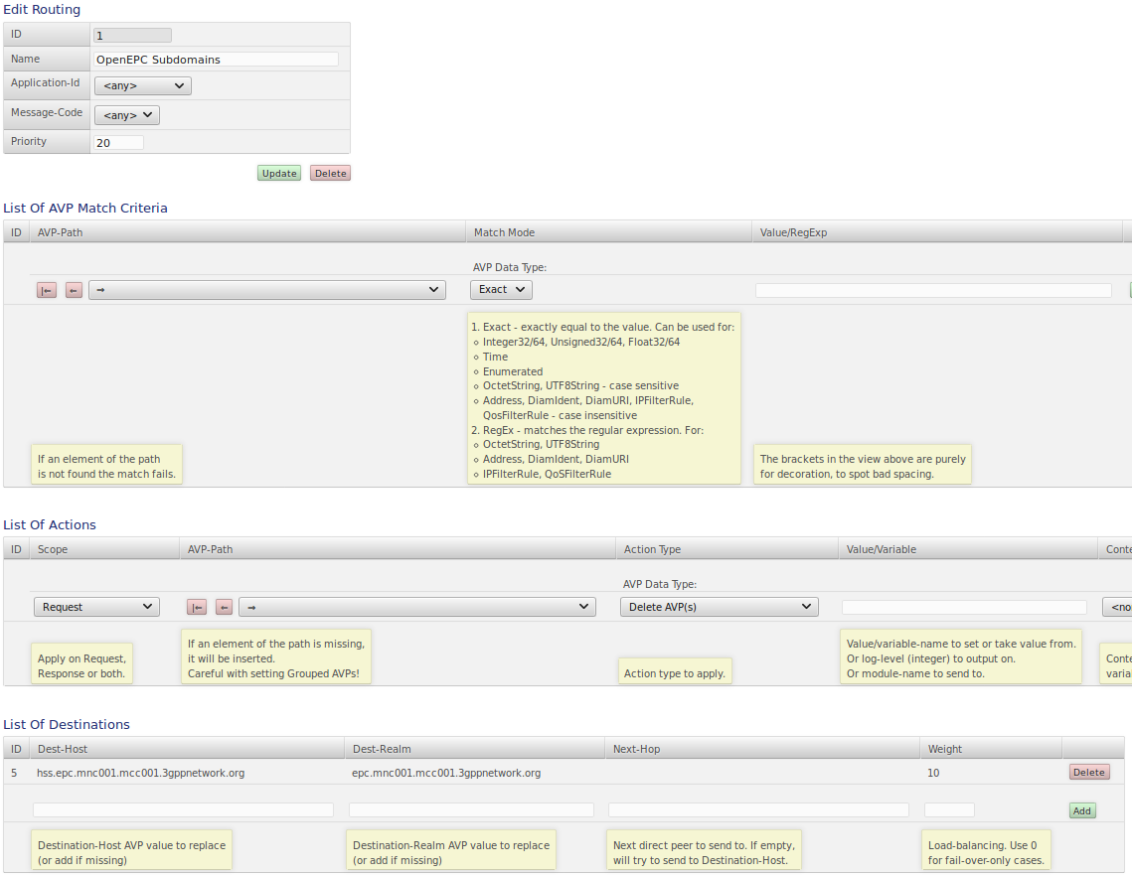


Fig. 4.5. Ruta genérica DRA

4.3. DESPLIEGUE OPERADOR ROJO EN KVM

El operador rojo se diferencia del verde principalmente en la configuración y aprovisionamiento del sistema. Para demostrar que ambos operadores son completamente independientes, se usarán subredes diferentes, además de modificar otros parámetros que identifican de manera única a cada operador. En el figura 4.6 se puede ver como es el despliegue del operador rojo.

En cada nodo, existe un fichero en la carpeta *install*, llamado data_topology_vars.sh. Todos los *scripts* de OpenEPC usan variables de entorno. Estas variables se encuentran todas en ese fichero. De esta manera se facilita adaptar todo el sistema a entornos diferentes.

En este fichero se tiene que modificar dos cosas:

- **IPs máquinas:** Se deben modificar todas las IPs por la que requiera el nuevo entorno. En este caso, se ha seguido la siguiente regla. Se mantiene el 4 octeto de la dirección IP y se modifica el 3 octeto sumándole diez, menos en la red de *managment* donde se le resta uno. De esta manera es más sencillo de trabajar con el segundo operador.
- **Identificadores Operador:** Cada red móvil de cada operador en el mundo se iden-

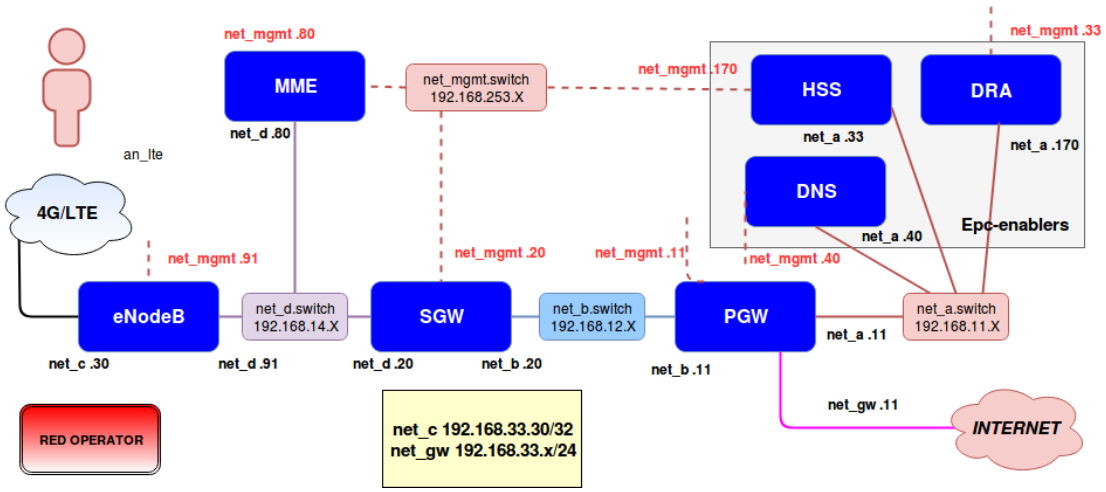


Fig. 4.6. Despliegue Operador Rojo

tifica de manera única e inequívoca mediante dos identificadores:

- **Mobile Country Code (MCC):** este código corresponde al país donde se encuentra la red del operador. En el caso de España este código es el 214.
- **Mobile Network Code (MNC):** este código identifica una red concreta dentro de un país. En el caso español, aunque solo existen 4 redes de telefonía físicas, los operadores móviles virtuales tienen su propio MNC.

El primer operador contaba con MCC 001 y MNC 001. Para reutilizar parte del aprovisionamiento que posteriormente se usará para la solución de *roaming*, los códigos seleccionados son MCC 001 y MNC 002. Estos códigos se pueden cambiar si se quiere. Posteriormente, se explicará cómo configurar el aprovisionamiento si se cambian estos códigos. Si se quiere realizar *roaming* en sentido inverso, también habrá que configurar el MNC visitado como 001 en el fichero.

Una vez que se haya modificado el fichero data_topology_vars.sh conformo al *setup* deseado, se ha de incluir en todas las máquinas y proceder a la reinstalación. Todos los ficheros de aprovisionamiento usan las variables del sistema. De esta manera todo quedara automáticamente configurado y aprovisionado dentro de OpenEPC.

Existe una excepción, el servicio DNS no se configura automáticamente usando dicho fichero, se tendrán que modificar todas las entradas. Para evitar realizar este proceso manualmente en cada reinstalación, se puede modificar el fichero pdns_data.sql.template en la carpeta *powerdns*. En ese fichero se encuentran todas las entradas DNS. Estas entradas DNS se añaden a la base de datos en el proceso de instalación usando el *script* de OpenEPC.

4.4. CONFIGURACIÓN Y APROVISIONAMIENTO DE ITINERANCIA SOBRE KVM

Tras tener desplegados ambos operadores funcionando correctamente, se ha de configurar la solución de itinerancia que se desarrolló en profundidad en el capítulo 3. En la Figura 4.7 se puede ver cómo quedaría el desligue completo.

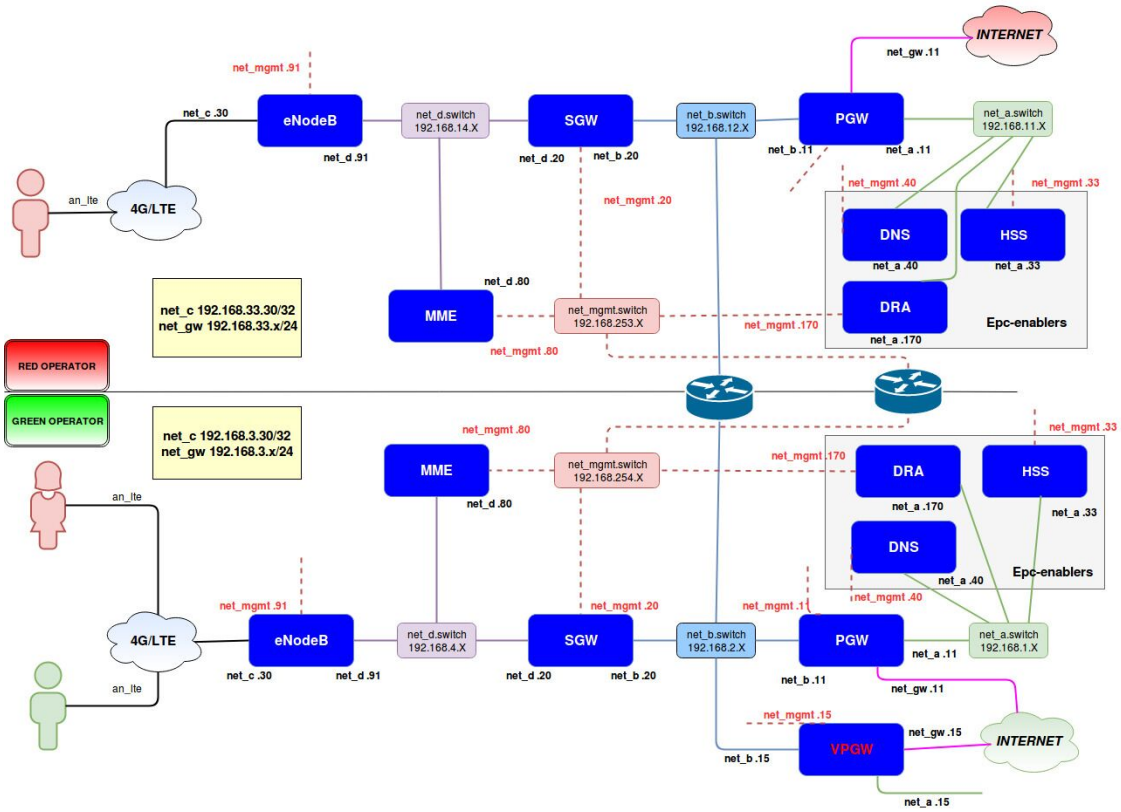


Fig. 4.7. Esquema completo del despliegue sobre KVM

En primer lugar se necesitará una máquina extra para el usuario en *roaming*. Será una máquina clonada del operador rojo para el usuario Alice. El usuario se conectará a la red del operador verde, que es el operador visitado. Se procederá exactamente igual que con las máquinas anteriores.

Se necesitará además comunicar ambos operadores. Como ya se ha analizado en el diseño de la solución, la señalización entre ambos operadores se realizara a través de los DRA/DRE. Además, para el caso de *home routed roaming* se usará una red de interconexión en el plano de datos.

En KVM, debido a la configuración del NAT, las redes de *managment* de ambos operadores están interconectadas por un *bridge*. Por tanto, ya existe comunicación entre ambos operadores. Para el caso de *home routed roaming*, se usará un *router* virtual basado en OpenWRT para realizar la interconexión, siendo la red de conmutación más simple que se puede hacer.

OpenWRT es una distribución de linux optimizada para funcionar en *router* físicos.

Tiene diferentes distribuciones para distintos tipos de *hardware* y además una genérica que permite crear *routers* virtuales usando máquinas virtuales. En los anexos se encontrará una guía de instalación para desplegar el *router* OpenWRT que se necesita y configurarlo.

En la solución *breakout roaming* se necesita una máquina clonada del PGW correspondiente al operador rojo. Esta máquina mantiene la misma configuración y solo se tendrán que modificar las interfaces e IPs correspondientes para que pertenezca a las subredes del operador verde. Téngase en cuenta que habrá que renombrar las interfaces tras clonar la máquina.

Por último, se ha de aprovisionar el sistema. Como se ha visto anteriormente el cambio entre una salida u otra para el tráfico, se realiza cambiando el DNS. Lo primero que se tendrá que hacer, será configurar las entradas DNS en el operador verde que identifican tanto el DRA del operador rojo, como el APN que usan los usuarios en el operador rojo. En la Figura 4.8 se ve la configuración de las diferentes entradas.

List Of Records

ID ↑↓	Name ↑↓	Type ↑↓	Content ↑↓	TTL ↑↓	
2101	epc.mnc002.mcc001.3gppnetwork.org	SOA	localhost. root.localhost. 2006101001 10800 900 3600 3600	3600	Delete
2102	epc.mnc002.mcc001.3gppnetwork.org	NS	ns.epc.mnc002.mcc001.3gppnetwork.org	3600	Delete
2103	ns.epc.mnc002.mcc001.3gppnetwork.org	A	192.168.11.40	3600	Delete
2104	dns.epc.mnc002.mcc001.3gppnetwork.org	A	192.168.11.40	3600	Delete
2105	vmhost.epc.mnc002.mcc001.3gppnetwork.org	A	192.168.253.1	3600	Delete
2106	gw.epc.mnc002.mcc001.3gppnetwork.org	A	192.168.33.1	3600	Delete
2107	aaa_proxy.epc.mnc002.mcc001.3gppnetwork.org	A	192.168.253.72	3600	Delete
2155	dra.epc.mnc002.mcc001.3gppnetwork.org	A	192.168.253.170	3600	Delete
2156	default.apn.epc.mnc002.mcc001.3gppnetwork.org	NAPTR	100 999 "s" "x-3gpp-pgw:x-s8-gtp" "" net-b-srv.pgw.node.epc.mnc002.mcc001.3gppnetwork.org	3600	Delete
2157	default.apn.epc.mnc002.mcc001.3gppnetwork.org	NAPTR	100 999 "s" "x-3gpp-pgw:x-s5-pimp" "" net-b-srv.pgw.node.epc.mnc002.mcc001.3gppnetwork.org	3600	Delete
2158	net-b-srv.pgw.node.epc.mnc002.mcc001.3gppnetwork.org	SRV	0 2123 net-b.pgw.node.epc.mnc002.mcc001.3gppnetwork.org	3600	Delete
2160	net-b.pgw.node.epc.mnc002.mcc001.3gppnetwork.org	A	192.168.2.15	3600	Delete
		<any>		60480	Add

Fig. 4.8. Entradas DNS en el operador verde

A parte de configurar el registro DNS referente al gw y el correspondiente al DRA, se han de añadir los registros DNS del APN. En este caso, se compone de los últimos cuatro registros. Los dos primeros corresponden a las dos posibles soluciones de tunelación implementadas, GTP y PIMP. Ambas apuntan al tercer registro que es de tipo SRV, que a su vez devolverá el cuarto registro con la IP correspondiente del PGW de salida. Para poder seleccionar que PGW se necesita en cada momento, se añadirá un quinto registro con la IP del V-PGW. Dinámicamente se modificará el registro SRV para que devuelva la IP del PGW o del V-PGW. Con este sistema, se es capaz de seleccionar el PGW que usará el usuario cuando se conecta a la red.

Para que esto funcione se requiere cambiar determinadas configuraciones relativas al APN. Como se vio anteriormente, el MME sustituye el APN-OI por el APN-OI del operador visitado, si el APN lo permite. En OpenEPC por defecto esta opción esta activada. Para modificarla se ha de desactivar la casilla en el operador rojo tal como se muestra en la figura 4.9.

Por último, se debe permitir a cada usuario conectarse a la red visitada. Por defecto,

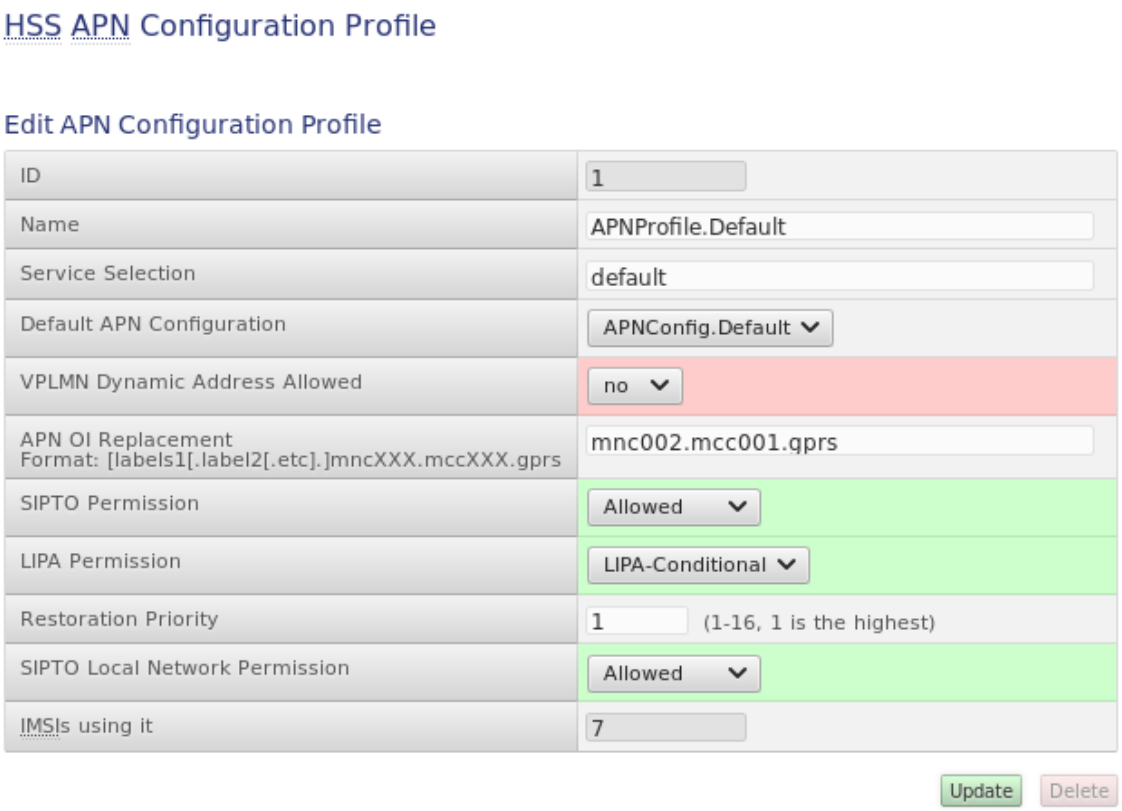


Fig. 4.9. Desactivación del direccionamiento dinámico del VPLMN

un usuario solo se puede conectar a aquellas redes que estén expresamente habilitadas. Si en el apartado anterior se han usado el Mobile Country Code 001 y el Mobile Network Code 002, la itinerancia está habilitada en todos los usuarios. En caso contrario, si se quiere personalizar el operador, se deberán seguir los siguientes pasos.

Primero se debe crear una nueva red visitada en la base de datos como se muestra en figura 4.10

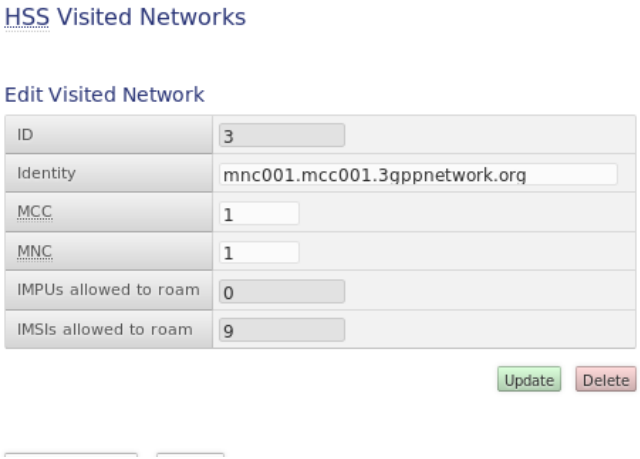


Fig. 4.10. Creación de nueva red visitada

Una vez creada, para cada uno de los usuarios que se les quiera permitir la itinerancia,

se debe ir al panel HSS International Mobile Subscriber Identifier (IMSI) y habilitar la red. Para ello en el apartado *Allowed Roaming* se añadirá la red creada anteriormente como se muestra en la figura 4.11.

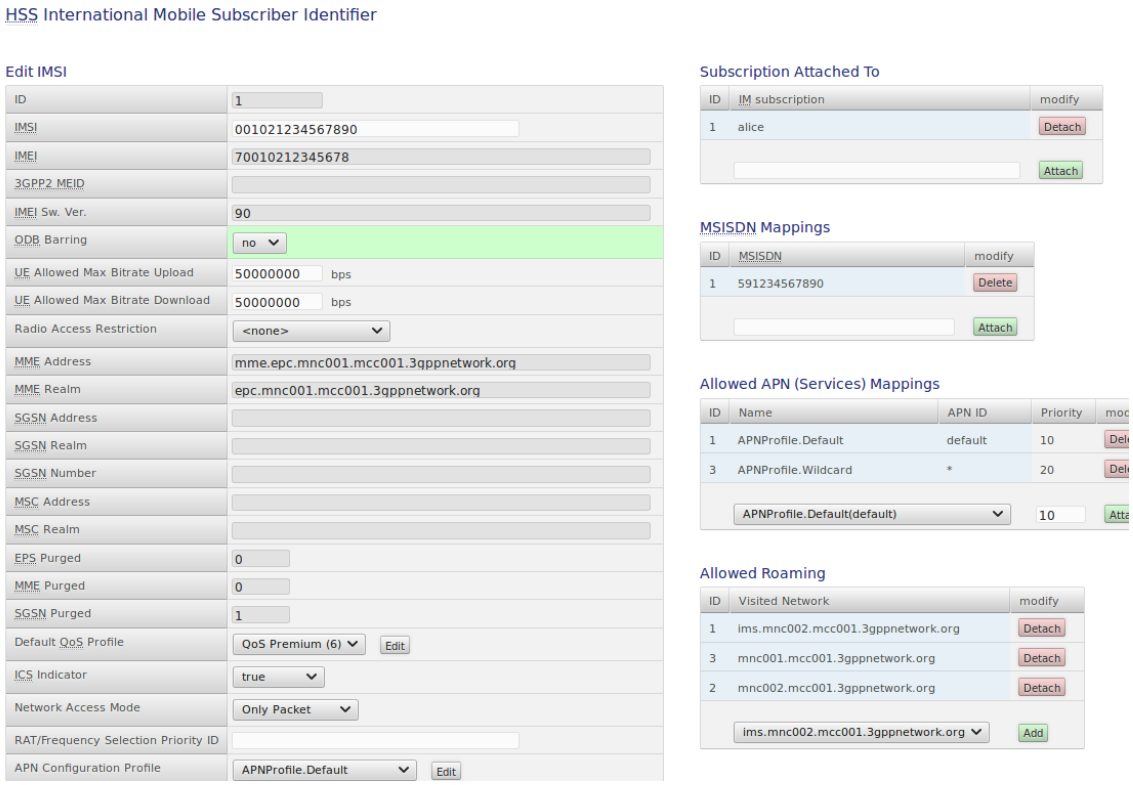


Fig. 4.11. Panel HSS International Mobile Subscriber Identifier (IMSI)

4.5. VALIDACIÓN DE LA SOLUCIÓN EN KVM

Para validar el correcto funcionamiento del despliegue se analizará la correcta conectividad de cada uno de los usuarios en cada uno de los escenarios. Se van a analizar tres aspectos clave:

- **Conectividad:** La primera prueba a realizar es la correcta conectividad del usuario con servidores que se encuentren en una red externa de paquetes. Se usará el bien conocido protocolo ICMP mediante el uso del *ping* a un servidor externo conectado a través de Internet
- **Punto de salida:** Puesto que se está analizando un escenario de *roaming*, es importante comprobar que el tráfico sale por el PGW correspondiente. Para ello se usará *traceroute* que permite identificar el primer salto del tráfico. De esta manera se puede identificar de manera inequívoca por donde sale el tráfico a internet.
- **TCP/IP:** Por último, se ha de comprobar que el usuario puede hacer uso correcto de TCP/IP. Para ello se usará *wget* y se descargará la página web de la universidad

www.uc3m.es. De esta manera se comprobará el correcto funcionamiento de los DNS y por tanto del protocolo UDP y además la correcta descarga de la página web que usa el protocolo de transporte TCP.

En primer lugar se va a validar el correcto despliegue del operador verde. Para ello se harán las comprobaciones comentadas anteriormente sobre el cliente Bob del operador verde. El resultado se puede observar en la figura 4.12.

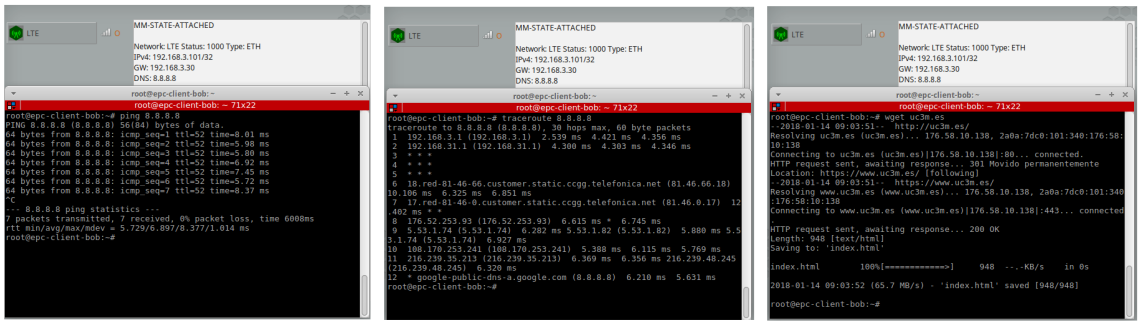


Fig. 4.12. Validación cliente Bob verde

Como se puede ver ha pasado las tres pruebas y se da por válido su correcto funcionamiento. Se repetirán los pasos anteriores para validar el operador rojo usando el cliente Bob del operador rojo. El resultado se encuentra en las figuras 4.13

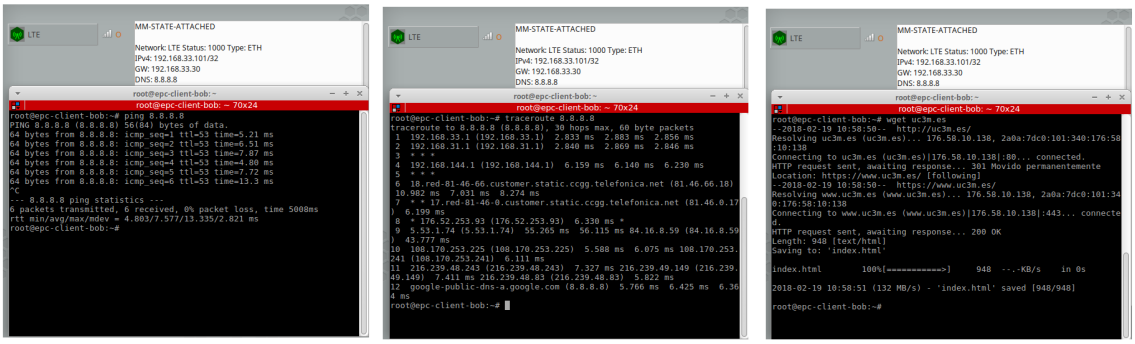


Fig. 4.13. Validación cliente Bob rojo

Las pruebas de validación también son correctas en este caso y se puede dar valido su funcionamiento, comprobando así que ambos operadores son funcionales.

El siguiente paso será validar el escenario de *roaming* con ambas soluciones. Se usará el usuario Alice perteneciente al operador rojo que está conectado a la red LTE del operador verde. Primero se validará que el usuario Alice funciona usando la solución *home routed*. El resultado se puede observar en la figura 4.14.

Se puede observar la correcta conectividad del usuario a través del PGW del operador rojo. Queda validado por tanto el primer supuesto de la solución. Se desconectará el

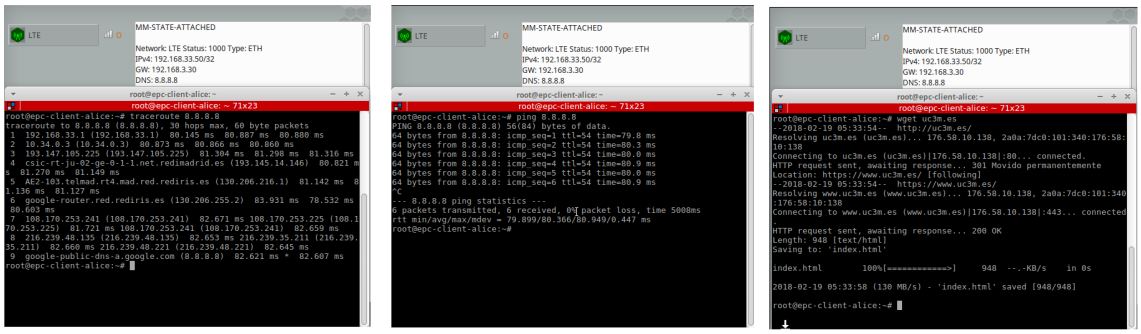


Fig. 4.14. Validación cliente Alice home routed roaming

cliente Alice de la red y se procederá a modificar la entrada DNS que permite seleccionar el V-PGW como salida al tráfico correspondiente al segundo supuesto *localbrekout*. El resultado se encuentra en la figura 4.15.

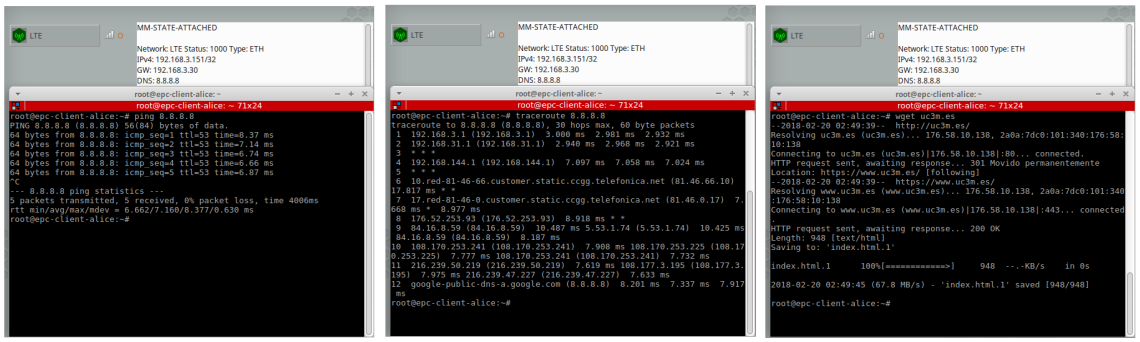


Fig. 4.15. Validación cliente Alice localbreakout roaming

El segundo supuesto ha pasado las pruebas de validación. Todos los elementos han pasado las pruebas de validación, se puede dar por válida la solución implementada sobre KVM. Como se puede apreciar la solución de itinerancia es viable en un despliegue virtualizado sobre KVM.

4.6. DESPLIEGUE SOLUCIÓN SOBRE OPENSTACK USANDO ORQUESTACIÓN

Se ha demostrado como solución propuesta funciona correctamente en entorno de pruebas realizado sobre KVM en una sola máquina. El siguiente paso será probar la solución en un entorno más parecido a un escenario real, con diferentes servidores alojados en distintos centros de cómputo. En este caso, la plataforma de virtualización usada será OpenStack.

Previamente se han analizado las ventajas e inconvenientes de esta plataforma que se tendrán en cuenta a la hora de realizar el despliegue. Como ya se comentó, el objetivo

principal del uso de esta plataforma es automatizar el despliegue usando orquestación. Se busca conseguir, que sin intervención humana, el orquestador que usa el proyecto 5GEx pueda desplegar ambos operadores e interactuar con ambas soluciones de *roaming*.

En los anexos se encuentra una guía de instalación de OpenStack donde se explica detalladamente como desplegar un *cluster* de OpenStack básico. Para desplegar este entorno se cuenta con servidores físicos profesionales. Cada uno de ellos tiene, un procesador Intel Xenon de 16 núcleos (32 virtualizados), 128 gigas de RAM y 5 Terabytes de disco duro.

Debido al alto coste de los servidores, el proyecto solo tiene 2 servidores físicos cuando el entorno necesita mínimo 4 equipos (2 por *cluster*), como se vio anteriormente. Para solucionar el problema, se instalará un *cluster* de OpenStack en cada servidor usando máquinas virtuales de KVM hasta obtener el número de equipos necesarios. Debido a la gran cantidad de recursos que necesita el operador virtualizado, un solo nodo de cómputo tendría un tamaño excesivo para hacer copias de seguridad. Se usarán dos nodos de cómputo en vez de uno para evitar tener el problema anterior.

Como referencia sobre la potencia necesaria, durante las pruebas se observó un consumo medio de 60 gigas de RAM y 24 procesadores virtualizados en cada uno de los operadores. A diferencia que en el caso anterior, donde una estación de trabajo portátil potente era suficiente, al añadir la capa de OpenStack, la estación de trabajo es insuficiente.

Una vez instalados ambos *clusters* con un *tenant* específico para cada operador y tras comprobar el correcto funcionamiento del sistema, se puede proceder con la instalación de ambos operadores.

4.6.1. CONSIDERACIONES PREVIAS

Durante el proceso de prueba se observó que OpenStack requiere una serie de ajustes adicionales para su correcto funcionamiento en el escenario propuesto. Como se vio anteriormente Neutron no está pensado para permitir que las máquinas virtuales redirijan tráfico dentro del *cluster*. Para que los nodos de OpenEPC puedan redirigir tráfico, tanto en OpenStack como en las máquinas virtuales hay que realizar diversas modificaciones.

Aunque el despliegue se puede realizar en dos *clusters* dentro del mismo servidor, para este proyecto se usará un entorno más complejo similar a un despliegue real. Cada uno de los servidores físicos estará ubicado en un centro de cómputo distinto. Ambos servidores están interconectados por una red de conmutación perteneciente al proyecto 5G Exchange. Esta red integra un plano de datos y un plano de control a los que se conectaran las interfaces correspondientes de los *clusters*. En la siguiente figura 4.16 se puede observar el entorno para el despliegue.

Para permitir que Heat, el orquestador de OpenStack, pueda generar todos los recursos necesarios se darán permisos de administrador a los *tenants*. OpenStack no permite

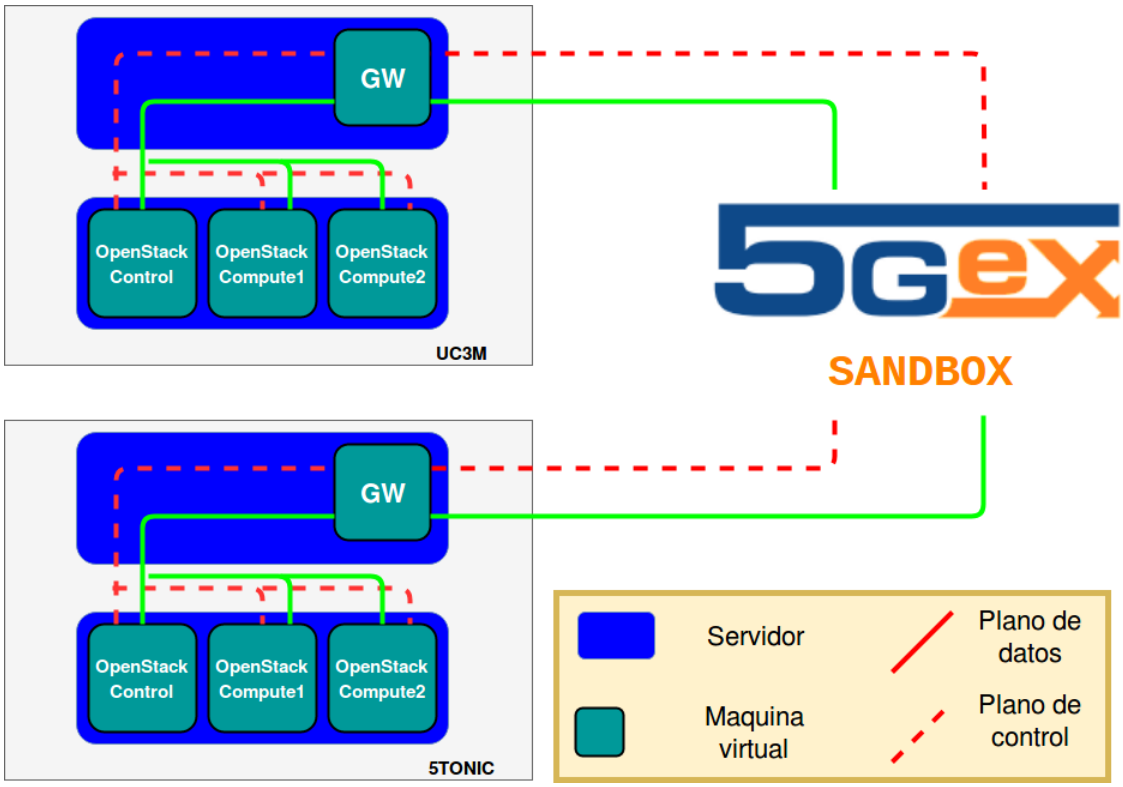


Fig. 4.16. Entorno físico para el despliegue

a usuarios normales crear IPs flotantes e IPs públicas con direccionamiento estático. Estas IPs han de ser estáticas y no dinámicas, puesto que la configuración de OpenEPC usa direccionamiento estático. Más adelante se detallarán los motivos por los cuales es necesario ambos tipos de IPs.

Dentro de OpenEPC los cambios serán menores. Consistirán en su mayoría en adecuar el direccionamiento IP al nuevo entorno que hemos usado y ajustar algunos parámetros correspondientes a las tarjetas *Ethernet*. Más adelante se explicará la razón de dichos cambios.

4.6.2. DIRECCIONAMIENTO IP

En el despliegue realizado sobre KVM, aunque se usaba direccionamiento IP privado, dentro de la misma máquina se comportaban como IP públicas enrutables. Sin embargo, en OpenStack las subredes son privadas sin conectividad al exterior independientemente del uso de uno o varios servidores. Cuando el tráfico tiene que salir del *cluster* se utiliza *nating* para tener conectividad con el exterior.

Durante las pruebas se descubrió que el uso del NAT suponía un problema a la hora de intercambiar paquetes entre operadores diferentes. Existen dos comunicaciones diferenciadas entre ambos operadores.

La primera de ellas corresponde a la señalización. Gracias al uso del DRA la situación

se simplifica enormemente. Solo hace falta comunicación entre ambos DRAs, por tanto solo se necesita una IP pública estática para cada DRA. OpenStack proporciona un mecanismo denominado IPs flotantes. Estas son IPs públicas asignadas por OpenStack que se pueden asociar a cualquier interfaz privada con conectividad a la red pública. Mediante NAT, permite comunicación con el exterior a través de esa interfaz.

El segundo flujo de comunicación corresponde al túnel GTP. En un principio se puede pensar en usar la solución anterior, pero a diferencia que DIAMETER, el protocolo GTP usa campos de la cabecera IP. Esto implica una incompatibilidad con el uso de NAT. La respuesta se genera en base a la cabecera GTP y no a la cabecera IP. Por tanto el PGW intenta enviar la respuesta a la IP privada del SGW con la que no tiene conectividad. La solución al problema consistió en eliminar la net_b y conectar las interfaces correspondientes directamente a la red pública. De esta manera, las interfaces del túnel GTP tiene IPs públicas. Tras estas modificaciones el entorno sobre OpenStack quedará modificado como se muestra en la figura 4.17.

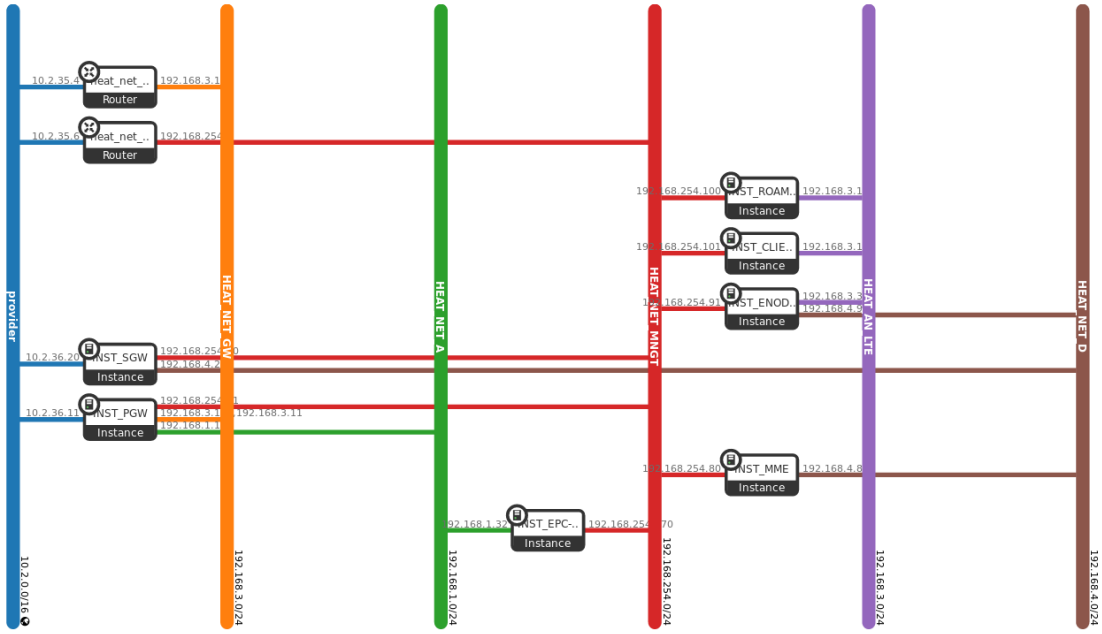


Fig. 4.17. Despliegue de red sobre OpenStack

4.6.3. CONFIGURACIÓN DE RED

Los cambios realizados en el direccionamiento IP necesariamente requieren una serie de cambios en la tabla de rutas de los distintos nodos. La red de conmutación de paquetes usada es la *sandbox* del proyecto 5G Exchange. La red está compuesta de diferentes islas a las que se le ha asignado un direccionamiento en concreto. Para este despliegue se están usando dos islas, la isla UC3M y la isla 5TONIC.

El direccionamiento asignado para ambas es el siguiente:

- **Isla UC3M:**
  - **Plano de datos:** 10.2.0.0/16
  - **Plano de control:** 172.17.3.0/24
- **Isla 5TONIC:**
  - **Plano de datos:** 10.34.0.0/16
  - **Plano de control:** 172.17.34.0/24

Los diferentes nodos de OpenStack tendrán asignada una dirección IP del plano de control. De esta manera el orquestador multidominio podrá interactuar con los diferentes *clusters*. Los nodos de OpenEPC son servicios de red. Aunque OpenEPC tenga una red de control, a efectos del despliegue global, esta pertenece a los servicios de red, por tanto el tráfico será enrutado por el plano de datos.

Al eliminar el *nating*, tanto SGW como PGW necesitan IPs públicas. Se ha asignado el rango 10.2.36.0/24 para asignar IPs a los nodos conectados a la net_b. Para asignar las diferentes IPs flotantes se le ha asignado a OpenStack un rango de IPs 10.2.35.0/24. En el caso del despliegue en 5TONIC los rangos son 10.34.35.0/24 y 10.34.35.0/24. Tanto en el PGW como en el SGW se tendrá que cambiar las IPs correspondientes en el fichero data_topology_vars.sh

Las interfaces virtuales en OpenStack, para reducir el gasto de recursos dentro de la misma máquina, deshabilitan el cálculo del *checksum* y la comprobación del mismo. De esta manera, el *checksum* solo se calcula cuando el tráfico ha de salir del *cluster*. Como se vio anteriormente, en OpenStack, el tráfico que generan los clientes dentro del *cluster* no existe como tal, puesto que no viene de una tupla IP/MAC conocida. Por tanto, cuando ese tráfico sale del *cluster* el *checksum* no es recalculado y los paquetes son descartados en el destino. Para solucionar esto, se ha forzado a los clientes a calcular el *checksum*. Para ello en el directorio /etc/udev/rules.d/ se ha añadido el fichero 50-eth_tso.rules:

```
1 SUBSYSTEM=="net", ACTION=="add", NAME="mgmt", RUN+="/sbin/ethtool -K
      mgmt tx off", RUN+="/sbin/ethtool -K mgmt tso off", RUN+="/sbin
     /ethtool -K mgmt gso off", RUN+="/sbin/ip link set dev mgmt mtu
     1300"
2 SUBSYSTEM=="net", ACTION=="add", NAME="an_lte", RUN+="/sbin/ethtool
     -K an_lte tx off", RUN+="/sbin/ethtool -K an_lte tso off", RUN+=
     "/sbin/ethtool -K an_lte gso off", RUN+="/sbin/ip link set dev
     an_lte mtu 1300"
```

Además, se ha reducido la MTU a 1300 para evitar la fragmentación de paquetes que se producía debido a la adición de las cabeceras del túnel GTP. La MTU también se modifica en el mismo fichero.

4.6.4. DESPLIEGUE OPERADORES CON HEAT

OpenStack tiene múltiples herramientas que son de utilidad a la hora de desplegar entornos complejos. El proyecto 5GEx requiere poder desplegar el entorno completo de manera automática. Para ello, se usara el moduló Heat de OpenStack del que ya se ha hablado antes.

Primero, se deberán implementar todos los cambios anteriormente expuestos en los diferentes nodos y crear sus respectivas imágenes. Posteriormente se subirán las imágenes correspondientes a OpenStack. El proceso se puede realizar tanto desde la consola de comandos como desde el *dashboard*. Se recomienda usar la terminal para agilizar el proceso.

```
openstack image create "epc-enablers" \
--file epc-enablers.qcow2 \
--disk-format qcow2 --container-format bare \
--public --project gilan
```

Una vez añadidos todos los recursos necesarios, se va a usar orquestación para desplegar los operadores aprovechando el módulo heat de OpenStack consiguiendo automatizar el proceso. Para ello, se ha de generar un fichero *yaml* definiendo la estructura del *stack* de cada operador. En los anexos se puede encontrar los ficheros completos creados para este proyecto. En las siguientes líneas se desarrollarán los puntos más importantes de estos ficheros.

Los ficheros de *yaml* tienen tres partes diferenciadas. La versión del *template*, una sección de parámetros y una sección de recursos. En la sección de paramentos se definen aquellos recursos que ya existen en OpenStack y se quieren usar.

```
        heat_template_version: 2014-10-16

        parameters:

        image_client_bob:
        type: string
        label: Image name or ID
        description: Image to be used for compute instance
        default: client-bob #CirrOS 0.3.5 DT uses "CentOS 7", img
            available in our openstack
        [...]

        keys:
        type: string
        description: the SSH keys to be installed on the servers
        default: GiLAN

        dns_nameserver:
        type: string
```

```
19          description: the IP address of YOUR local/global dns
                nameserver.
20          default: "8.8.8.8"
21
22          public_network: #the public network to be used
23          type: string
24          label: Public network name or ID
25          description: Public network
26          default: provider # Name of public network, needs to be
                external and already exist in OpenStack
```

Para el despliegue de los operadores estos paramentos serán las imágenes, la red pública y las claves ssh. Para usar uno de estos parámetros en el momento de generar un recurso se usará la siguiente sintaxis:

```
1  {get_param: nombre_del_parametro}
```

Por último esta la tercera sección que incluye la definición de todos los recursos. Los recursos son aquellos elementos que no existen en OpenStack desplegados y son parte de *stack* del operador. Primero se han de crear las redes, subredes y los *router* necesarios.

```
1   resources:
2
3   # NETWORKS
4   heat_net_mgmt:
5   type: OS::Neutron::Net
6   properties:
7   name: HEAT_NET_MNGT
8   [...]
9   # SUBNETS
10  heat_net_mgmt_subnet: #creating the subnets
11   type: OS::Neutron::Subnet
12   depends_on: heat_net_mgmt
13   properties:
14    network_id: { get_resource: heat_net_mgmt }
15    cidr: "192.168.254.0/24"
16    gateway_ip: "192.168.254.2"
17    dns_nameservers:
18      - { get_param: dns_nameserver }
19  [...]
20  #ROUTERS
21  heat_net_mgmt_router:
22  type: OS::Neutron::Router
23  depends_on: heat_net_mgmt_subnet
24  properties:
25    name: heat_net_mgmt_router
26    external_gateway_info:
27    network: { get_param: public_network }
```

Hay que destacar dos aspectos importantes sobre las líneas anteriores. En primer lugar el campo *depends_on* permite secuenciar el despliegue de todos los elementos. Esto evitará que OpenStack intente crear una subred si no existe la red primero. Posteriormente, será especialmente útil para asegurar que los nodos del operador se despliegan en orden. El otro campo importante es *gateway_ip*. Las máquinas de OpenEPC tienen configurado como *gateway* una dirección diferente a la habitual. Para que el entorno funcione correctamente, el *gateway* configurado en OpenStack deberá coincidir con el de OpenEPC.

Una vez creada la topología de red, han de crearse las diferentes interfaces. Se usarán tres tipos de interfaces diferentes, interfaces públicas para *router*, interfaces con IPs privadas o IPs públicas, e interfaces con una IP flotante asociada.

```
1    resources:
2
3    # INTERFACES
4
5    #ROUTER PUBLIC INTERFACE
6     public-interface: #creating the router interface, on an existing
          router
7     type: OS::Neutron::RouterInterface
8     depends_on: heat_net_mgmt_router
9     properties:
10      router_id: { get_resource: heat_net_mgmt_router }
11      subnet: { get_resource: heat_net_mgmt_subnet }
12    [...]
13
14    #VM INTERFACES
15     heat_pgw_net_b_port:
16     type: OS::Neutron::Port
17     properties:
18       network_id: { get_param: public_network }
19       mac_address: 52:54:00:9d:da:8a
20       port_security_enabled: false
21       fixed_ips:
22        - ip_address: 10.2.36.11
23
24     heat_pgw_net_a_port:
25     type: OS::Neutron::Port
26     depends_on: heat_net_a_subnet
27     properties:
28       network_id: { get_resource: heat_net_a }
29       mac_address: 52:54:00:56:1d:1d
30       port_security_enabled: false
31       fixed_ips:
32        - ip_address: 192.168.1.11
33
34     #INTERFACE WITH FLOATING IP
35
36      heat_epc-enablers_net_mgmt_port:
```

```
      type: OS::Neutron::Port
      depends_on: heat_net_mgmt_subnet
        properties:
          network_id: { get_resource: heat_net_mgmt }
          mac_address: 52:54:00:38:d5:16
          port_security_enabled: false
          fixed_ips:
      #       - ip_address: 192.168.254.30 #default
      #       - ip_address: 192.168.254.32 #dashboard
      #       - ip_address: 192.168.254.33 #hss
      #       - ip_address: 192.168.254.40 #dns
      #       - ip_address: 192.168.254.45
      #       - ip_address: 192.168.254.46
      #       - ip_address: 192.168.254.47
      #       - ip_address: 192.168.254.70
              - ip_address: 192.168.254.170 #dra_ip

      floating_ip_epc:
      type: OS::Neutron::FloatingIP
      properties:
        floating_network: {get_param: public_network}
        port_id: { get_resource: heat_epc-enablers_net_mgmt_port }
        floating_ip_address: 10.2.35.170
```

Para poder crear IPs públicas flotantes estáticas, el *tenant* deberá tener permisos de administrador. Si no fuera así, no se podría fijar una IP pública estática, aunque si se podría asignar una IP dinámica. Se puede observar, que todas las interfaces tienen el *port security* desactivado. Esto permite que se reenvié todo el tráfico, evitando filtrar aquellos paquetes cuyo par IP/MAC no conozca OpenStack. En OpenEPC el traficó de usuario en la net_gw tiene esas características. Además, al deshabilitar *port security* no se necesita indicarle a OpenStack las múltiples IPs que tiene *epc-enablers*. Por último se crearán las diferentes instancias que alojaran cada uno de los servicios.

```
    # SERVERS/INSTANCES
    heat_client_bob:
    type: OS::Nova::Server
    depends_on: [heat_client_bob_net_mgmt_port,
       heat_client_bob_net_an_lte_port, heat_enodeb ]
    properties:
     image: { get_param: image_client_bob }
     name: INST_CLIENT_BOB
     flavor: { get_param: flavor }
     key_name: { get_param: keys }
     networks:
      - port: {get_resource: heat_client_bob_net_mgmt_port}
      - port: {get_resource: heat_client_bob_net_an_lte_port}
```

Una vez preparados ambos ficheros se puede desplegar el *stack* tanto desde el *dash-*

board como desde la terminal de comandos. En ambos casos se recomienda activar la opción *enable rollback*. En caso de fallo al crear el *stack*, con esta opción se borrará automáticamente todo lo que se haya creado, dejando constancia de donde se produjo el error.

```
openstack stack create -t fichero.yaml op1_gilan --enable-rollback
```

4.6.5. DESPLIEGUE CLIENTE ALICE EN ROAMING

Para poder realizar las diversas pruebas de *roaming* se necesitará un cliente perteneciente al operador rojo, conectado a la red LTE del operador verde. Este cliente es el mismo que en el correspondiente al despliegue de KVM, el cliente Alice. El despliegue de este cliente se va a realizar de manera independiente al despliegue de los operadores. De esta manera posteriormente se podrá añadir tantos clientes como sean necesarios para trabajos futuros.

El cliente Alice que se desplegara deberá incluir el fichero 50-eth_tso_.rules configurando correctamente la MTU y desactivando la optimización del *checksum*, al igual que se hizo con el cliente Bob en el apartado anterior.

Una vez, que tenemos la máquina configurada correctamente se ha de subir al *cluster* correspondiente al operador verde.

```
openstack image create "client_alice_2" \
--file client_alice_2.qcow2 \
--disk-format qcow2 --container-format bare \
--public --project gilan
```

El fichero XML para desplegar el cliente es ligeramente diferente al mostrado anteriormente durante los despliegues de los operadores. Cuando se despliega el cliente, a diferencia que en el despliegue de los operadores, los *routers*, redes y subredes ya están configuradas. Por tanto en el fichero XML, los elementos que ya están previamente creados, han de referenciarse como parámetros en vez de como recursos. En los anexos se puede encontrar el fichero XML completo.

4.6.6. MULTI-TENANT: V-PGW

En este punto se ha desplegado correctamente ambos operadores y la solución *home-routed roaming* deberá funcionar correctamente antes de proceder a desplegar la solución alternativa conocida como *localbreakout*. Para ello como se comentó antes, se ha de desplegar una copia del PGW del operador rojo en el centro de cómputo del operador verde, cerca del usuario en itinerancia.

Conceptualmente, el operador verde solamente cede sus recursos de cómputo. La ad-

ministración, configuración y mantenimiento del V-PGW corre a cargo del operador rojo, al igual que los costes asociados al despliegue del mismo. Por ese motivo, se plantea crear un nuevo *tenant* en el *cluster* perteneciente al operador verde totalmente independiente de este. De esta manera el operador rojo, podrá desplegar este V-PGW de manera aislada del resto de la red y se tendrá un mejor control de los recursos consumidos.

Una vez más, la escasez de IPs públicas entraña un problema. Tanto la red de net_b como la red net_gw no sufren cambios respecto al escenario anterior. A efectos del sistema, existe solamente un cambio de localización física, se seguirá usando el túnel S8 ya que se trata de un PGW externo.

El cambio más significativo lo encontramos en la red de control. Como se ha comentado anteriormente, el control del nodo corresponde al operador rojo. Dado que el PGW no se encuentra en la misma subred se deberán usar IPs públicas para interconectar el V-PGW con los servicios de control del operador rojo. En este caso, se puede usar IPs flotantes al igual que se usaron para interconectar ambos DRAs.

En el operador rojo tendremos que configurar la entrada DNS, correspondiente al V-PGW con su IP flotante correspondiente, mientras que en el virtual PGW tendremos que configurar IP publica del DNS del operador rojo. Para el escenario propuesto con estas modificaciones es suficiente. Si se quisiera usar mayor cantidad de señalización para servicios extra, se debería incluir un DRA en el mismo *tenant* que el V-PGW. De esta manera el intercambio de señalización se realizara igual que en el caso de *roaming* pero dentro del mismo operador.

Por último, como elemento opcional, se ha modificado el rango de IPs privadas ofrecidas a los clientes. Este cambio se ha realizado para hacer más visual el punto de salido del cliente. En un despliegue real, usar el mismo rango de IPs privadas no tendría ningún impacto.

4.6.7. DESPLIEGUE CLIENTES ADICIONALES

En el escenario propuesto por el proyecto 5GEx cuando se realiza el cambio al V-PGW, los clientes que estén usando la red en ese momento no se verán afectados por el cambio. Solo usaran el V-PGW los nuevos clientes conectados. Por tanto, se ha de comprobar que durante el cambio entre soluciones de *roaming*, los clientes conectados no pierden conectividad. De la misma manera, cuándo se decida volver a la solución *home-routed roaming*, los clientes que estén usando el V-PGW seguirán conectados a este nodo. Dicho nodo no se eliminará hasta que todos los clientes conectados a él dejen de usar el servicio.

Para poder comprobar que el funcionamiento del sistema cumple con lo indicado anteriormente, se necesita más de un cliente en itinerancia. OpenEPC viene aprovisionado con 5 clientes en cada operador. Hasta ahora solo se han usado dos clientes Alice y Bob. Para poder realizar dichas pruebas se necesitan al menos dos clientes en itinerancia adi-

cionales. En este caso se usaran Charlie y Dave. Opcionalmente se puede añadir un quinto cliente conocido como Emma.

Para crear un nuevo cliente se ha de clonar el cliente Alice en itinerancia. La imagen clonada ha de configurarse correctamente como un cliente diferente. Para ello en primer lugar, se ha de modificar el fichero XML. Nótese, que en este caso el fichero que se ha de modificar no es el del servicio mm.xml.

Durante la instalación del cliente, se realiza un enlace simbólico entre mm.xml y el fichero de configuración real mm_vmteam_alice.xml. Por tanto, se renombrará el fichero con el nombre del cliente correspondiente, y dentro del mismo se sustituirá el International Mobile Subscriber Identity (IMSI) que identifica a Alice, por el correspondiente al cliente que se quiere instalar.

Antes de reinstalar el servicio, se han de volver a renombrar las interfaces para actualizar las direcciones MAC. Una vez hecho esto, se puede reinstalar el cliente, indicando en el *script* el nombre del nuevo. De esta manera se generará el enlace simbólico entre el nuevo fichero y el mm.xml.

4.7. VALIDACIÓN DE LA SOLUCIÓN EN OPENSTACK

Al igual que en el caso de KVM, para validar el correcto funcionamiento del despliegue se analizará la correcta conectividad de cada uno de los usuarios en cada uno de los escenarios. Para ello se analizarán los mismos tres aspectos clave:

- **Conectividad:** La primera prueba a realizar es la correcta conectividad del usuario con servidores que se encuentren en una red externa de paquetes. Se usará el bien conocido protocolo ICMP mediante el uso del *ping* a un servidor externo conectado a través de internet.
- **Punto de salida:** Puesto que se está analizando un escenario de *roaming*, es importante comprobar que el tráfico sale por el PGW correspondiente. Para ello se usará *traceroute* que permite identificar el primer salto del tráfico. De esta manera se puede identificar de manera inequívoca por donde sale el tráfico a internet.
- **TCP/IP:** Por último, se ha de comprobar que el usuario puede hacer uso correcto de TCP/IP. Para ello se usará *wget* y se descargará la página web de la universidad www.uc3m.es. De esta manera se comprobará el correcto funcionamiento de los DNS y por tanto del protocolo UDP y además la correcta descarga de la página web que usa el protocolo de transporte TCP.

Primero se ha de validar la correcta automatización del despliegue. Se eliminarán todos los *stacks* y se volverá a desplegar desde cero ambos operadores, los clientes y el V-PGW. Si tras este procedimiento se pasan todas las pruebas de validación, el modelo estará correctamente configurado para la automatización.

En primer lugar se desplegaran ambos operadores y los clientes. Se va a validar primero el correcto despliegue del operador verde en OpenStack. Para ello se harán las comprobaciones comentadas anteriormente sobre el cliente Bob del operador verde.

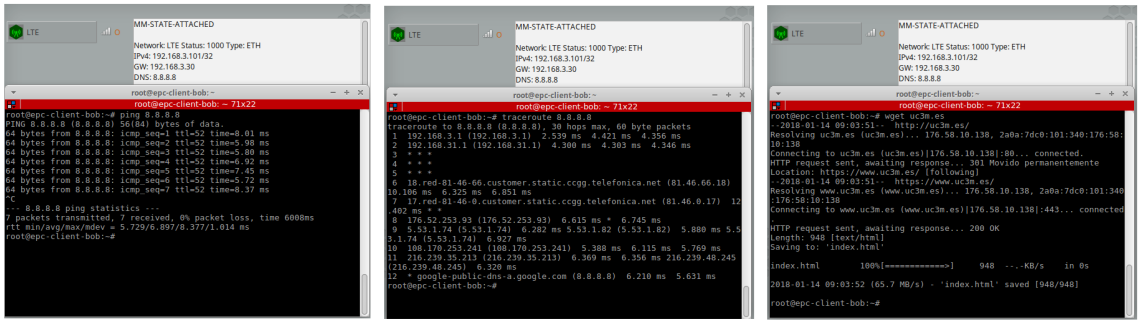


Fig. 4.18. Validación cliente Bob verde

Como se puede ver ha pasado las tres pruebas y se da por válido su funcionamiento. Se repetirán los pasos anteriores para validar el operador rojo usando el cliente Bob del operador rojo. Las pruebas de validación también son correctas en este caso y se puede

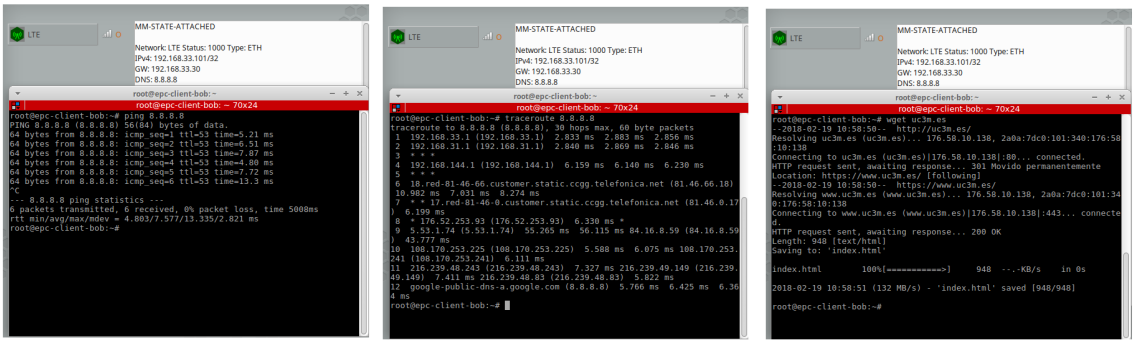


Fig. 4.19. Validación cliente Bob rojo

dar válido su funcionamiento.

Para validar el escenario de *roaming* se usará el usuario Alice perteneciente al operador rojo que está conectado a la red LTE del operador verde. Como primer paso se validará que el usuario funciona usando la solución *home routed*.

Queda validado el primer supuesto de la solución. Nótese la existencia de latencia elevada, debido a que el tráfico pasa por el conmutador del 5G-PPP localizado en Pisa, Italia. Se pasará a desplegar el V-PGW y a modificar la entrada DNS que permite seleccionar el V-PGW como salida al tráfico correspondiente al segundo supuesto *localbrekout*. Comprobaremos que el nuevo cliente Charlie se conecta a través del V-PGW realizando las tres pruebas de validación. El cliente Alice seguirá conectado por el PGW del operador rojo.

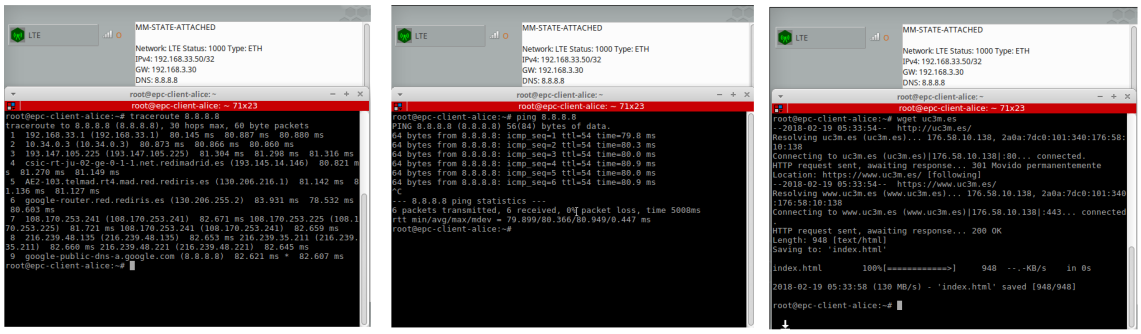


Fig. 4.20. Validación cliente Alice home routed roaming

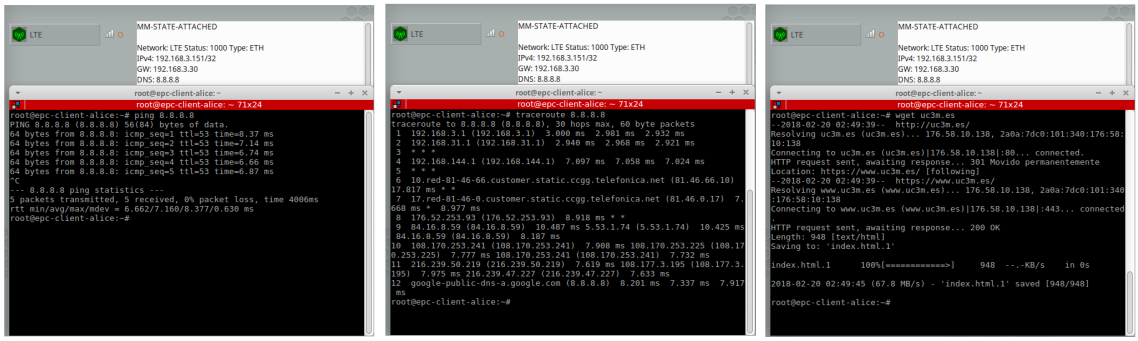


Fig. 4.21. Validación cliente Charlie localbreakout roaming

El segundo supuesto ha pasado las pruebas de validación y el cliente Alice no ha perdido su conectividad. Todos los elementos han pasado las pruebas de validación, se puede dar por válida la solución implementada sobre OpenStack. Como se puede apreciar la solución de itinerancia es viable en un despliegue virtualizado sobre OpenStack. Además podemos concluir que el entorno está correctamente configurado para su despliegue automático.

4.8. RENDIMIENTO DEL DESPLIEGUE DEL V-PGW SOBRE OPENSTACK

El despliegue del V-PGW tiene un coste asociado. A la hora de desplegarlo existen dos posibles casos. En el primer caso, se considera que el V-PGW nunca ha sido instanciado. El coste asociado en este caso es mayor, ya que se tendrá que transmitir la imagen y crear los recursos asociados. En el segundo caso, si el V-PGW ya ha sido instanciado puede ser que aun exista una copia en cache. Si es así, como en OpenStack, se ahorran recursos a la vez que se disminuye el tiempo de despliegue. Se han realizado medidas obteniendo un tiempo medio de despliegue en el segundo escenario de 2 minutos y medio.

5. CONCLUSIONES Y TRABAJOS FUTUROS

5.1. CONCLUSIONES

El objetivo de este trabajo fin de grado ha sido analizar la solución que propone el proyecto europeo 5G Exchange y realizar una implementación independiente para validar su viabilidad tecnológica. El resultado ha sido exitoso tanto en el despliegue y la validación. Durante el proceso se ha descubierto que las tecnologías de virtualización usadas no escalan correctamente.

En cuanto a KVM, solo ha sido usado para facilitar el despliegue por etapas del sistema, no se buscó una escalabilidad sobre esta plataforma. OpenStack es muy usado en la industria para alojar servidores que funcionan como *end-points*. La configuración de red en OpenStack es poco flexible para maximizar la seguridad y la robustez en esos escenarios. El *core* de red virtual incluye máquinas que enrutan tráfico. Esto es un elemento diferencial respecto a los servidores comunes.

En el proyecto se han podido solucionar todos los problemas relativos a la configuración de OpenStack, consiguiendo un entorno funcional. Sin embargo, para entornos de producción se necesitan alternativas preparadas para virtualizar funciones de red, termino más conocido en inglés como Network Function Virtualization (NFV). Además para gran cantidad de tráfico, esto debería complementarse con soporte para redes definidas por software, conocido en inglés como Software Defined Networks (SDN).

La solución aquí propuesta es funcional aunque se recomienda como futuros trabajos, el uso de plataformas de virtualización adaptadas que aun están en fase de desarrollo. En cualquier caso, la solución propuesta es viable a medio plazo, lo que corresponde con el margen que la unión europea ha dado las operadoras antes de fijar definitivamente los precios mayoristas.

5.2. TRABAJOS FUTUROS

- En cuanto a la solución de *roaming* se podría trabajar en los siguientes aspectos:
  - Automatizar el despliegue del V-PGW usando orquestación para que este solo funcione si se le está dando uso. Esto evitaría desperdiciar recursos en los centros de cómputo.
  - Realizar el cambio entre ambas soluciones de *roaming* de manera dinámica atendiendo a criterios de eficiencia y económicos.
- Sobre el despliegue de ambos operadores virtualizados:

- Integrar el *core* de Red con un acceso radio LTE físico, ya que en el despliegue actual, tanto usuarios como acceso radio están simulados.
- Proporcionar acceso radio *untrusted* 3GPP mediante WIFI usando el módulo ePDG para ofrecer un acceso radio no simulado que no requiera de licencia para poder experimentar con él.
- Despliegue de la herramienta *Flowmoon* para monitorizar en tiempo real el tráfico cursado.
- Configurar y provisionar el soporte para tunelación PIMP.
- Configurar y provisionar Policy Control Resource Function (PCRF) para permitir el cobro a los usuarios por los servicios prestados.

BIBLIOGRAFÍA

[1] D.-G. for the Information Society y M. (European Comission), "Flash eurobarometer 454, The end of roaming charges within the EU", European Comision, Report, sep. de 2017.

[2] 5G-PPP. (2017). Interviews about key achievements of 5GEX project, [En línea]. Disponible en: https://www.5gex.eu/wp/?p=653.

[3] EEE. (2015). Reglamento (UE) 2015/2120 del parlamento europeo y del consejo, [En línea]. Disponible en: http://www.boe.es/doue/2015/310/L00001-00018.pdf.

[4] UIT, *Sobre la union internacional de telecomunicaciones (UIT)*, 2018. [En línea]. Disponible en: https://www.itu.int/es/about/Pages/default.aspx.

[5] 3GPP. (2018). About 3GPP, [En línea]. Disponible en: http://www.3gpp.org/about-3gpp.

[6] ——, (2010). Release 9, [En línea]. Disponible en: http://www.3gpp.org/specifications/releases/71-release-9.

[7] ——, (2017). Release 14, [En línea]. Disponible en: http://www.3gpp.org/release-14.

[8] Telefonica, S.A. (ene. de 2017). Telefonica selects huawei to build large scale virtual EPC network in 13 countries as part of its unica program. inglés, Telefonica, S.A., [En línea]. Disponible en: https://www.telefonica.com/documents/737979/133242187/PR+TEF_Huawei_vEPC+Network_Final.pdf/21cd5fdd-45ca-4bcf-9f40-8b59322fe896?version=1.0.

[9] Core Network Dynamics. (2017). OpenEPC, [En línea]. Disponible en: http://www.openepc.com/.

[10] OpenStack. (2018). What is OpenStack?, [En línea]. Disponible en: https://www.openstack.org/software/.

[11] Huawei Technologies Co., Ltd. (2017). Lte international roaming whitepaper, [En línea]. Disponible en: http://carrier.huawei.com/en/technical-topics/core-network/lte-roaming-whitepaper.

[12] The CentOS Project. (2017). Centos linux, [En línea]. Disponible en: https://www.centos.org/download/.

ANEXO GUÍA DE INSTALACIÓN KVM

Para instalar KVM primero se debe comprobar si el host tiene soporte *hardware* para la virtualización. Se abrirá una terminal y se ejecutará la siguiente línea:

```
$ egrep -i 'vmx|svm' --color=always /proc/cpuinfo
```

Si los campos relativos a la CPU contienen *vmx* o *svm*, el host es adecuado para la instalación de KVM.

INSTALACIÓN KVM AND QEMU

Se necesitara instalar los siguientes paquetes:

```
sudo apt-get install qemu-kvm libvirt-bin  bridge-utils
```

INSTALACIÓN VIRT MANAGER

Si el host cuenta con interfaz gráfica se puede instalar una herramienta que permite gestionar máquinas virtuales y redes.

```
sudo apt-get install virt-manager
```

En caso de que el host no cuente con interfaz gráfica, se puede instalar Virt Manager en otro equipo y conectarse en remoto al host para gestionar y/o visualizar las diferentes máquinas virtuales.

ANEXO GUÍA DE INSTALACIÓN OPENWRT

OpenWRT es una distribución de linux pensada para funcionar en *router* físicos sustituyendo el *firmware* de fábrica. Esta distribución cuenta con una versión que se puede correr en máquinas virtuales, permitiendo crear *router* virtualizados con múltiples opciones de configuración.

En el caso de este proyecto se está usando la distribución LEDE, para x86 en 64 bits. Esta distribución tiene drivers para interfaces de tipo *virtio*. Al crear la máquina virtual se deberán seleccionar interfaces virtuales de ese tipo.

CONFIGURACIÓN EN EL ROUTER

Toda la configuración relativa a redes se encuentra en el fichero /config/network. Se adjunta un ejemplo de ese fichero con la configuración usada en el entorno de KVM.

```

config interface 'net_b'
        option type lan
        option ifname 'eth0'
        option proto 'static'
        option ipaddr '192.168.2.1'
        option netmask '255.255.255.0'

config interface 'net_b_2'
        option type lan
        option ifname 'eth1'
        option proto 'static'
        option ipaddr '192.168.12.1'
        option netmask '255.255.255.0'

config route
        option interface 'lan'
        option target '192.168.2.0'
        option netmask '255.255.255.0'
        option gateway '192.168.2.1'

config route
        option interface 'lan'
        option target '192.168.12.0'
        option netmask '255.255.255.0'
        option gateway '192.168.12.1'
```

Para que estas reglas funcionen correctamente se deberá desactivar el *firewall*. No obstante también se puede configurar el *firewall* adecuadamente si se quiere un control

más finos del tráfico.

```
etc/init.d/firewall disable
etc/init.d/firewall stop
```

ANEXO GUÍA DE INSTALACIÓN OPENSTACK

En este proyecto se cuenta con un único servidor físico. Para desplegar un *cluster* de OpenStack se usarán dos máquinas virtuales, una para el nodo de cómputo y otra para el nodo de control. En esta guía se usará el sistema operativo Centos7[12]. En primer lugar se han de preparar ambas máquinas sobre KVM.

CONFIGURACIÓN SUBREDES

```
# virsh net-define /root/examplenetwork.xml
# virsh net-start examplenetwork
# virsh net-autostart examplenetwork
```

Se necesitarán dos subredes:

```
<network>
<name>openstack_net</name>
<uuid>f610f325-106a-b9b8-7b95-dcb0e13e7281</uuid>
<bridge name='virbr0' stp='on' delay='0'/>
<mac address='52:54:00:11:14:5b'/>
<ip address='10.0.1.0' netmask='255.255.255.0'>
</ip>
</network>
```

```
<network>
<name>openstack_mgmt</name>
<uuid>f610f325-1a6a-b9b8-7b95-dbb0e13e7281</uuid>
<bridge name='virbr1' stp='on' delay='0'/>
<mac address='52:54:00:11:14:7c'/>
<ip address='10.0.0.1' netmask='255.255.255.0'>
</ip>
</network>
```

DEFINICIÓN MAQUINAS VIRTUALES

Para crear las máquinas virtuales se puede usar el siguiente *script*:

```
#!/bin/bash -
set -o nounset                  # Treat unset variables as an error

virt-install \
--name centos7_rm_controller \
```

```
--ram 16384 \
--disk path=./centos7_rm_controller.qcow2,size=128 \
--vcpus 8 \
--os-type linux \
--os-variant rhel7 \
--network bridge=virbr0 \
--graphics none \
--console pty,target_type=serial \
--location 'http://mirror.i3d.net/pub/centos/7/os/x86_64/' \
--extra-args 'console=ttyS0,115200n8 serial'
```

Tras la creación de las máquinas virtuales se debe instalar Centos7. Se recomienda usar una instalación mínima para ambos nodos.

GESTIÓN DE INTERFACES

Para añadir o modificar interfaces se podrá usar la Virt Manager o la terminal. Para crearlas:

```
sudo virsh attach-interface --domain centos7_rm_compute --source
    virbr2 --type bridge --model virtio --persistent
```

Para destruirlas:

```
sudo virsh detach-interface --domain centos7_rm_compute --type
    bridge --mac 52:54:00:fd:ef:b1 --persistent
```

Además, se cambiará el nombre a las interfaces modificando el fichero:

```

/etc/udev/rules.d/70-persistent-ipoib.rules

#Estructura de la entrada
ACTION=="add", SUBSYSTEM=="net", ATTR{address}=="52:54:00:72:6c:da",
    NAME="mgmt"
ACTION=="add", SUBSYSTEM=="net", ATTR{address}=="52:54:00:b7:a3:88",
    NAME="net"
```

NESTED VIRTUALIZATION

Para que OpenStack pueda usar el hypervisor de KVM, se necesita habilitar la virtualización anidada. Esta tiene que estar soportada por el host. Se comprueba ejecutando el siguiente comando en el host:

```
```

```
sudo virt-host-validate
```

Si todo esta correcto debería aparecer todos los campos como PASS. Puede darse el caso, que no esté habilitada la virtualización anidada, en cuyo caso aparecerá el siguiente *output*:

```
QEMU: Checking if IOMMU is enabled by kernel
                                    : WARN (IOMMU appears to be
   disabled in kernel. Add intel_iommu=on to kernel cmdline
   arguments)
```

Para habilitar se ha de modificar grub en /etc/default/grub, y añadir la siguiente línea:

```
GRUB_CMDLINE_LINUX= intel_iommu=on
```

Tras la reinstalación, se puede volver a comprobar si todos los campos aparecen como PASS. Si todo es correcto se ha de habilitar esta opción en las máquinas virtuales previamente creadas. Esto se realiza editando el fichero de la máquina desde la terminal:

```
sudo virsh edit NOMBRE_MAQUINA
```

Se tiene que añadir la siguiente línea:

```
<cpu mode=?custom? match=?exact?>
<model fallback=?allow?>SandyBridge</model>
<feature policy=?require? name=?vmx?/> #hay que añadir esta linea
</cpu>
```

Tras realizar *shutdown* de la máquina virtual, se puede comprobar que se ha configurado todo correctamente ejecutando:

```
 egrep -c '(vmx|svm)' /proc/cpuinfo
```

Si el *output* es distinto de cero, se podrá usar el hypervisor de KVM en OpenStack.

INSTALACIÓN OPENSTACK

Para instalar OpenStack se seguirá la guía de instalación proporcionada por OpenStack para la versión Ocata. Para el despliegue propuesto será necesario instalar *Neutron* en su opción 2 *self-service networks* y el módulo opcional *Heat* para poder usar orquestación. Antes de comenzar con la instalación se ha de desactivar SELINUX, puesto que entra en conflicto con Neutron.

https://docs.openstack.org/ocata/install-guide-rdo/index.html

ACCESO AL DASHBOARD

Para poder acceder al *dahsboard* y a las máquinas virtuales desde un equipo monitor se han de crear dos tuneles ssh. Uno para el acceso al *dashboard* y otro para el acceso a las máquinas.

El acceso al *dashboard* se hará con el siguiente túnel:

```
ssh user@host -L 8080:controller_guest_default_net_ip:80
```

Para acceder a las máquinas se necesitará otro túnel:

```
ssh user@host -L 6080:controller_guest_default_net_ip:6080
```

Se abrirá un navegador en el equipo monitor y se accederá al *dashboard* a través de localhost:8080/dashboard. Cuando se quiera acceder a una máquina en concreto, se deberá abrir la consola en pantalla completa y sustituir en la url *controller* por *localhost*. De esta manera se podrá trabajar con las diferentes máquinas.

ANEXO FICHEROS DE CONFIGURACIÓN OPENEPC

EPC-ENABLERS: DRA

```
<?xml version="1.0" encoding="UTF-8"?>
<!--

  OpenEPC 7 - CGF Configuration File

  For full documentation on the parameter please check:
    - wharf/cfg/wharf.xml for the core configuration
    - wharf/modules/module_name/module_name.xml for the
       configuration of each individual modules

-->

<Wharf path="%OpenEPC_path_wharf">

  <Core>
    <Debug log="3" memory="6" memory_status="1"/>
    <SharedMemory kbytes="16384" />
    <ProcessMemory kbytes="4096" />
    <WorkerPool count="2" queue_size="8" />
  </Core>

  <Module binaryFile="modules/console/console.so" >
    <![CDATA[
      <WharfConsole>
        <Prompt text=" DRA Console >"/>
        <Acceptor type="tcp" port="%dra_mgmt_console_port" bind="%
           dra_mgmt_ipv4" />
        <Acceptor type="udp" port="%dra_mgmt_console_port" bind="%
           dra_mgmt_ipv4" />
      </WharfConsole>
    ]]>
  </Module>

  <Module binaryFile="modules/mysql/mysql.so" />

  <Module binaryFile="modules/cdp2/cdp2.so">
    <![CDATA[
      <DiameterPeer
        FQDN="%dra_diameter_fqdn"
        Realm="%base_realm"
        Vendor_Id="45569"
        Product_Name="OpenEPC-7c"
        AcceptUnknownPeers="1"
```

```
          DropUnknownOnDisconnect="1"
          Tc="30"
          Workers="4"
          QueueLength="32"
          TransactionTimeout="5"
          SessionsHashSize="128">

          <Peer FQDN="%hss_diameter_fqdn" Realm="%base_realm" port="%
              hss_mgmt_diameter_port"/>
          <Peer FQDN="%mme_diameter_fqdn" Realm="%base_realm" port="%
              mme_mgmt_diameter_port"/>
          <Peer FQDN="dra.%visited_realm" Realm="%visited_realm" port
              ="%dra_mgmt_diameter_port"/>

          <Acceptor bind="%dra_mgmt_ipv4" port="%
              dra_mgmt_diameter_port"  />

          <Auth id="16777216" vendor="10415"/><!-- 3GPP Cx -->
          <Auth id="16777216" vendor="4491"/><!-- CableLabs Cx -->
          <Auth id="16777216" vendor="13019"/><!-- ETSI/TISPAN Cx -->
          <Auth id="16777217" vendor="10415"/><!-- 3GPP Sh -->
          <Auth id="16777280" vendor="10415"/><!-- 3GPP Sp - not
              assigned yet -->
          <Auth id="16777251" vendor="10415"/><!-- 3GPP S6a/S6d -->
          <Auth id="16777251" vendor="0"/>    <!-- 3GPP S6a/S6d -->
          <Auth id="16777265" vendor="10415"/><!-- 3GPP SWx -->
        </DiameterPeer>
      ]]>
    </Module>

    <Module binaryFile="modules/dra/dra.so">
      <![CDATA[
        <DRA>
          <Database
            hostname="%dra_db_host"
            database="%dra_db_name"
            username="%dra_db_user"
            password="%dra_db_pass"/>
        </DRA>
      ]]>
    </Module>

    <Module binaryFile="modules/dra_standalone_breakout/
        dra_standalone_breakout.so" />

  </Wharf>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!--

  OpenEPC 7 - HSS Configuration File

  For full documentation on the parameter please check:
    - wharf/cfg/wharf.xml for the core configuration
    - wharf/modules/module_name/module_name.xml for the
       configuration of each individual modules

-->

<Wharf path="%OpenEPC_path_wharf">

  <Core>
    <Debug log="3" memory="6" memory_status="1"/>
    <SharedMemory kbytes="16384" />
    <ProcessMemory kbytes="4096" />
    <WorkerPool count="2" queue_size="8" />
  </Core>

  <Module binaryFile="modules/console/console.so">
    <![CDATA[
    <WharfConsole>
      <Prompt text=" HSS Console >"/>
      <Acceptor type="tcp" port="%hss_mgmt_console_port" bind="%
         hss_mgmt_ipv4" />
    </WharfConsole>
    ]]>
  </Module>

  <Module binaryFile="modules/sctp/sctp.so"/>

  <Module binaryFile="modules/cdp2/cdp2.so">
    <![CDATA[
      <DiameterPeer
        FQDN="%hss_diameter_fqdn"
        Realm="%base_realm"
        Vendor_Id="10415"
        Product_Name="CDiameterPeer"
        AcceptUnknownPeers="1"
        DropUnknownOnDisconnect="1"
        Tc="30"
        Workers="4"
        QueueLength="32"
        TransactionTimeout="5"
        SessionsHashSize="128">

```

```
47          <Peer FQDN="%icscf_diameter_fqdn"       Realm="%ims_realm"
                port="%icscf_mgmt_diameter_port"/>
48          <Peer FQDN="%scscf_diameter_fqdn"       Realm="%ims_realm"
                port="%scscf_mgmt_diameter_port"/>
49
50          <Peer FQDN="%andsf_diameter_fqdn"       Realm="%base_realm"
                port="%andsf_mgmt_diameter_port"/>
51          <Peer FQDN="%pcrf_diameter_fqdn"        Realm="%base_realm"
                port="%pcrf_mgmt_diameter_port"/>
52
53          <Peer FQDN="%dra_diameter_fqdn"         Realm="%base_realm"
                port="%dra_mgmt_diameter_port"/>
54
55          <Acceptor port="3868" bind="%hss_mgmt_ipv4"/>
56
57          <Auth id="16777216" vendor="10415"/><!-- 3GPP Cx -->
58          <Auth id="16777216" vendor="4491"/> <!-- CableLabs Cx -->
59          <Auth id="16777216" vendor="13019"/><!-- ETSI/TISPAN Cx -->
60          <Auth id="16777217" vendor="10415"/><!-- 3GPP Sh -->
61          <Auth id="16777280" vendor="10415"/><!-- 3GPP Sp - not
                assigned yet -->
62          <Auth id="16777280" vendor="45569"/><!-- CND Sp - not
                assigned yet -->
63          <Auth id="16777251" vendor="10415"/><!-- 3GPP S6a/S6d -->
64          <Auth id="16777265" vendor="10415"/><!-- 3GPP SWx -->
65          <Auth id="16777312" vendor="10415"/><!-- 3GPP S6c -->
66
67        </DiameterPeer>
68      ]]>
69    </Module>
70
71    <Module binaryFile="modules/mysql/mysql.so"/>
72
73    <Module binaryFile="modules/hss/hss.so">
74      <![CDATA[
75        <HSS>
76          <Database
77            hostname="%hss_db_host"
78            database="%hss_db_name"
79            username="%hss_db_user"
80            password="%hss_db_pass" />
81          <Provisioning
82            mcc="%mcc"
83            mnc="%mnc_short"
84            msin_prefix="4242"
85            imsi_digits="15"
86            >
87            <DefaultNewIMSIVisitedNetwork id="1" />
88            <DefaultNewIMSIVisitedNetwork id="2" />
89            <DefaultNewIMSIVisitedNetwork id="3" />
90
```

```
            <DefaultNewIMSIAllowedAPNProfile id="1" priority="10" />
            <DefaultNewIMSIAllowedAPNProfile id="2" priority="20" />
            <DefaultNewIMSIAllowedAPNProfile id="3" priority="30" />

            <DefaultNewIMSIValue column="id_imsu"
                                                 value="1" />
            <DefaultNewIMSIValue column="ue_ambr_upload"
                                       value="150000000" />
            <DefaultNewIMSIValue column="ue_ambr_download"
                                     value="150000000" />
            <DefaultNewIMSIValue column="id_access_restriction"
                                  value="NULL" />
            <DefaultNewIMSIValue column="apn_oi_replacement"
                                    value="%pgw_fqdn" />
            <DefaultNewIMSIValue column="id_qos_profile_default"
                                 value="1" />
            <DefaultNewIMSIValue column="id_apn_configuration_profile"
                               value="1" />
          </Provisioning>
          <Cx
            enabled="0"
            eventsLookupInterval="10"
            eventsMaxActionsAtOnce="10"
            userProfileXSDPath="%OpenEPC_path_wharf/modules/hss_cx/
               CxDataType_Rel12.xsd" />
          <Sh
            enabled="0"
            eventsLookupInterval="10"
            eventsMaxActionsAtOnce="10"
            userProfileXSDPath="%OpenEPC_path_wharf/modules/CHeSS_Sh/
               ShDataType_Rel12.xsd" />
          <Sp enabled="0"
            eventsLookupInterval="10"
            eventsMaxActionsAtOnce="10" />
          <S6ad
            enabled="1"
            eventsLookupInterval="10"
            eventsMaxActionsAtOnce="10" />
          <SWx
            enabled="0"
            eventsLookupInterval="10"
            eventsMaxActionsAtOnce="10" />
          <S6c
            enabled="0"/>
        </HSS>
      ]]>
    </Module>

    <Module binaryFile="modules/hss_s6ad/hss_s6ad.so"/>

    <Module binaryFile="modules/gsm_l3/gsm_l3.so"/>
```

```
133
134     <Module binaryFile="modules/pcsc/pcsc.so" />
135
136     <Module binaryFile="modules/usim/usim.so">
137       <![CDATA[
138         <USIM>
139           <Simulation
140             IMSI="%imsi_alice"
141             K="00000000000000000000000000000000"
142             OP="00000000000000000000000000000000"
143             SQN="000000000000"
144           />
145           <CardProfile
146             Name="Generic U/C/I-SIM"
147             Type="generic"
148             ATR="3B9F96801FC78031E073FE211B640750120082900084"
149             PIN1="1234"
150             ADM_PIN="x3332323133323332">
151           </CardProfile>
152           <CardProfile
153             Name="USIMERA Prime"
154             Type="usimera"
155             ATR="3B9E96801FC78031E073FE211B66D000750209007E"
156             PIN1="x31313131ffffffff"
157             ADM_PIN="x1111111111111111">
158             <Hardcoded Name="msisdn_alpha_len" Value="20"/>
159           </CardProfile>
160           <CardProfile
161             Name="Sysmocom USIM-gr1"
162             Type="osmocom"
163             ATR="3B9F94801FC78031E073FE211357122911020100 00C2"
164             PIN1=""
165             ADM_PIN="x3332323133323332">
166             <Hardcoded Name="msisdn_alpha_len" Value="14"/>
167             <Hardcoded Name="smsp_alpha_len" Value="12"/>
168           </CardProfile>
169           <CardProfile
170             Name="Sysmocom USIM-gr1.1"
171             Type="osmocom"
172             ATR="3B9F95801FC78031E073FE211357122911020100 00C2"
173             PIN1=""
174             ADM_PIN="x3332323133323332">
175             <Hardcoded Name="msisdn_alpha_len" Value="14"/>
176             <Hardcoded Name="smsp_alpha_len" Value="12"/>
177           </CardProfile>
178           <CardProfile
179             Name="Sysmocom USIM-SJS1"
180             Type="osmocom_sjs1"
181             ATR="x3B9F96801FC78031A073BE2113674320071800 0001A5"
182             PIN1="x30393132"
183             ADM_PIN="86459275">
```

```
            <Hardcoded Name="msisdn_alpha_len" Value="8"/>
            <Hardcoded Name="smsp_alpha_len" Value="6"/>
          </CardProfile>
          <CardProfile
            Name="Viasma"
            Type="viasma"
            ATR="x3B9F94801FC78031E073FE211357868506869842 18AB"
            PIN1="1234"
            ADM_PIN="88888888">
            <Hardcoded Name="set_pin_secret" Value="x90BF8E2EEF23F82E
                "/>
            <Hardcoded Name="msisdn_alpha_len" Value="14"/>
            <Hardcoded Name="smsp_alpha_len" Value="12"/>
          </CardProfile>
          <CardProfile
            Name="CND_Blue_USIM1"
            Type="cnd_blue_usim1"
            ATR="3B9F94801FC78031E073FE211357122911020100 00C3"
            PIN1="1234"
            ADM_PIN="x3233373631373939">
            <Hardcoded Name="msisdn_alpha_len" Value="14"/>
            <Hardcoded Name="smsp_alpha_len" Value="12"/>
          </CardProfile>
          <CardProfile
            Name="GreenCard"
            Type="grcard"
            ATR="x3B9C94801FC78031E073FE211B544F0109AD08"
            PIN1="1234"
            ADM_PIN="22222222">
            <Hardcoded Name="set_pin_secret" Value="x90BF8E2EEF23F82E
                "/>
            <Hardcoded Name="msisdn_alpha_len" Value="14"/>
            <Hardcoded Name="smsp_alpha_len" Value="13"/>
          </CardProfile>
          <CardProfile
            Name="Smartjac - nano/micro"
            Type="smartjac"
            ATR="x3B9F96C00A3FC6A08031E073FE211B65D001740F50810F6D"
            PIN1="1234"
            ADM_PIN="x41444D3131313131">
            <Hardcoded Name="msisdn_alpha_len" Value="24"/>
            <Hardcoded Name="smsp_alpha_len" Value="24"/>
            <Hardcoded Name="impu_len" Value="255"/>
            <Hardcoded Name="pin_adm_4" Value="x41444D3434343434"/>
            <Hardcoded Name="pin_unknown" Value="x35363738FFFFFFFF"/>
          </CardProfile>
        </USIM>
      ]]>
    </Module>
  </Wharf>
```

PGW

```
<?xml version="1.0" encoding="UTF-8"?>
<!--

  OpenEPC 7 - PGW Configuration File

  For full documentation on the parameter please check:
    - wharf/cfg/wharf.xml for the core configuration
    - wharf/modules/module_name/module_name.xml for the
       configuration of each individual modules

-->

<Wharf path="%OpenEPC_path_wharf">

  <Core>
    <Debug log="3" memory="6" memory_status="1"/>
    <SharedMemory kbytes="16384" />
    <ProcessMemory kbytes="4096" />
    <WorkerPool count="2" queue_size="8" />
  </Core>

  <Module binaryFile="modules/console/console.so">
    <![CDATA[
      <WharfConsole>
        <Prompt text=" PGW Console>"/>
        <Acceptor type="tcp" bind="%pgw_mgmt_ipv4"  port="%
           pgw_mgmt_console_port" />
      </WharfConsole>
    ]]>
  </Module>

  <Module binaryFile="modules/lma/lma.so">
    <![CDATA[
      <WharfLMA
        hash_size="32"
        seamless="0"
        enforce_s6b="no"
        enforce_pcef="no">
        <UserPlane
          ipv4="%pgw_net_b_ipv4"
          ipv6="" />
        <S5S8
          cp_ipv4="%pgw_net_b_ipv4"
          cp_ipv6="" />
      </WharfLMA>
    ]]>
  </Module>

```

```
47      <Module binaryFile="modules/gw_bindings/gw_bindings.so">
48        <![CDATA[
49          <GW_BINDINGS>
50              <Database
51                hostname="%pgw_db_gw_bindings_host"
52                database="%pgw_db_gw_bindings_name"
53                username="%pgw_db_gw_bindings_user"
54                password="%pgw_db_gw_bindings_pass"/>
55          </GW_BINDINGS>
56        ]]>
57      </Module>
58
59      <Module binaryFile="modules/mysql/mysql.so"/>
60
61      <Module binaryFile="modules/pdn_ops/pdn_ops.so">
62        <![CDATA[
63          <WharfPDN_Operations>
64            <Database
65              hostname="%pgw_db_pdn_ops_host"
66              database="%pgw_db_pdn_ops_name"
67              username="%pgw_db_pdn_ops_user"
68              password="%pgw_db_pdn_ops_pass"/>
69          </WharfPDN_Operations>
70        ]]>
71      </Module>
72
73      <Module binaryFile="modules/mobileip/mobileip.so">
74        <![CDATA[
75          <MobileIPPeer>
76            <Communicator type="udp_ipv4" default_local_addr="%
                pgw_net_b_ipv4" local_port="%pgw_net_b_pmip_port"/>
77          </MobileIPPeer>
78        ]]>
79      </Module>
80
81      <Module binaryFile="modules/gtp/gtp.so">
82        <![CDATA[
83          <GTP>
84            <Acceptor id="S5S8-GTP-C" type="udp" bind="%pgw_net_b_ipv4"
                port="%pgw_net_b_gtpc_port" />
85          </GTP>
86        ]]>
87      </Module>
88
89      <Module binaryFile="modules/gtp_intf/gtp_intf.so" />
90
91      <Module binaryFile="modules/routing_pgw/routing_pgw.so">
92        <![CDATA[
93          <WharfROUTING arp_enabled="yes" hash_size="32" buffer_size
              ="6000000">
94            <PathManagement
```

```
            hash_table_size="16"
            expires_interval="90"
            t3_response="3"
            n3_requests="3"
            check_interval="1"
           />
      </WharfROUTING>
      ]]>
    </Module>

    <Module binaryFile="modules/routing_encap/routing_encap.so" />

    <Module binaryFile="modules/routing_gtpu/routing_gtpu.so" />

    <Module binaryFile="modules/routing_raw/routing_raw.so" />

    <Module binaryFile="modules/cdp2/cdp2.so">
      <![CDATA[
        <DiameterPeer
          FQDN="%pgw_diameter_fqdn"
          Realm="%base_realm"
          Vendor_Id="45569"
          Product_Name="OpenEPC-7c"
          AcceptUnknownPeers="1"
          DropUnknownOnDisconnect="1"
          Tc="30"
          Workers="1"
          QueueLength="32"
          TransactionTimeout="5"
          SessionsHashSize="128">

          <Acceptor bind="%pgw_mgmt_ipv4" port="%
              pgw_mgmt_diameter_port" />


          <!-- <Auth id="16777272" vendor="10415"/> 3GPP S6b -->
          <!--  <Auth id="16777272" />              3GPP S6b -->
          <Auth id="16777238" vendor="10415"/> <!-- 3GPP Gx -->
          <Auth id="16777238" />               <!-- 3GPP Gx -->
          <Auth id="4"/>                       <!-- 3GPP Gy/Ro -->
          <Acct id="3"/>                       <!-- 3GPP Gz/Rf -->

        </DiameterPeer>
      ]]>
    </Module>

    <Module binaryFile="modules/client_s6b/client_s6b.so"/>
    <Module binaryFile="modules/client_gx/client_gx.so"/>
    <Module binaryFile="modules/client_ro/client_ro.so"/>
    <Module binaryFile="modules/client_rf/client_rf.so"/>

```

```
    <Module binaryFile="modules/pcef/pcef.so" >
      <![CDATA[
        <pcef
          hash_table_size="32">
          <Network
            mcc="%mcc"
            mnc="%mnc"
            release="12"
            extension="OCS"
            />
          <RatingGroup
            rating_group_id="1"
            cc_time="0"
            cc_currency_code="0"
            cc_money_value="0"
            cc_total_octets="0"
            cc_input_octets="1000"
            cc_output_octets="1000"
            cc_service_specific_units="0"
            />
        </pcef>
      ]]>
    </Module>

    <Module binaryFile="modules/bbf/bbf.so"/>

    <Module binaryFile="modules/routing_pcef/routing_pcef.so">
      <![CDATA[
        <Routing_PCEF>
          <Charging reporting_interval="20"/>
        </Routing_PCEF>
      ]]>
    </Module>


  </Wharf>
```

SGW

```
<?xml version="1.0" encoding="UTF-8"?>
<!--

  OpenEPC 7 - SGW Configuration File

  For full documentation on the parameter please check:
    - wharf/cfg/wharf.xml for the core configuration
    - wharf/modules/module_name/module_name.xml for the
       configuration of each individual modules

-->

<Wharf path="%OpenEPC_path_wharf">
  <Core>
    <Debug log="3" memory="6" memory_status="1"/>
    <SharedMemory kbytes="16384" />
    <ProcessMemory kbytes="4096" />
    <WorkerPool count="2" queue_size="8" />
  </Core>

  <Module binaryFile="modules/console/console.so">
    <![CDATA[
      <WharfConsole>
        <Prompt text=" SGW Console>"/>
        <Acceptor type="tcp" port="%sgw_mgmt_console_port" bind="%
            sgw_mgmt_ipv4" />
      </WharfConsole>
    ]]>
  </Module>

  <Module binaryFile="modules/mysql/mysql.so" />

  <Module binaryFile="modules/mobileip/mobileip.so">
    <![CDATA[
      <MobileIPPeer>
        <Communicator type="udp_ipv4" default_local_addr="%
            sgw_net_b_ipv4" local_port="%sgw_net_b_pmip_port"/>
      </MobileIPPeer>
    ]]>
  </Module>

  <Module binaryFile="modules/mag/mag.so">
    <![CDATA[
      <WharfMAG
        default_apn="%default_apn_short"
        ipcan="5"
        rat="1000"
        ipv4="%sgw_net_b_ipv4"
```

```
        ipv6=""
        lifetime="300"
        hash_size="32"
        s5s8_protocol="gtp">
        <Database
          hostname="%sgw_db_mag_host"
          database="%sgw_db_mag_name"
          username="%sgw_db_mag_user"
          password="%sgw_db_mag_pass"/>
        <DefaultLMA
          ipv4="%sgw_net_b_ipv4"
          ipv6=""/>
      </WharfMAG>
    ]]>
  </Module>

  <Module binaryFile="modules/mag_sgw/mag_sgw.so">
    <![CDATA[
      <MAG_SGW>
        <Control
          ipv4="%sgw_net_d_ipv4"
          ipv6="" />
        <UserPlane
          ipv4="%sgw_net_d_ipv4"
          ipv6="" />
      </MAG_SGW>
    ]]>
  </Module>

  <Module binaryFile="modules/routing/routing.so" >
    <![CDATA[
      <WharfROUTING>
        <Extension
          id="0"
          src_table="teid"
          mod_name="routing_gtpu"
          ipv4="%sgw_net_d_ipv4"
          ipv6=""/>
        <Extension
          id="1"
          mod_name="routing_gtpu"
          dst_table="teid"
          ipv4="%sgw_net_b_ipv4"
          ipv6="" />
        <PathManagement
          hash_table_size="16"
          expires_interval="90"
          t3_response="3"
          n3_requests="3"
          check_interval="1" />
      </WharfROUTING>
```

```
      ]]>
    </Module>

    <Module binaryFile="modules/routing_encap/routing_encap.so" />

    <Module binaryFile="modules/routing_gtpu/routing_gtpu.so" />

  <!--
    <Module binaryFile="modules/qos/routing_qos.so"/>
  -->

  <!--
    <Module binaryFile="modules/routing_pcc_sdf/routing_pcc_sdf.so" >
      <![CDATA[
        <Routing_PCC_SDF>
          <Charging
            octet_threshold="65535"
            packet_threshold="65535"
            time_threshold="120"/>
        </Routing_PCC_SDF>
      ]]>
    </Module>
  -->

    <Module binaryFile="modules/gtp/gtp.so">
      <![CDATA[
        <GTP>
          <Acceptor id="S11S4-GTP-C" type="udp" port="%
              sgw_net_d_gtpc_port" bind="%sgw_net_d_ipv4" />
          <Acceptor id="S5S8-GTP-C" type="udp" port="%
              sgw_net_b_gtpc_port" bind="%sgw_net_b_ipv4" />
        </GTP>
      ]]>
    </Module>

    <Module binaryFile="modules/gtp_intf/gtp_intf.so" />

    <Module binaryFile="modules/gw_bindings/gw_bindings.so" >
      <![CDATA[
        <GW_BINDINGS>
          <Database
            hostname="%sgw_db_gw_bindings_host"
            database="%sgw_db_gw_bindings_name"
            username="%sgw_db_gw_bindings_user"
            password="%sgw_db_gw_bindings_pass"/>
        </GW_BINDINGS>
      ]]>
    </Module>

    <Module binaryFile="modules/cdp2/cdp2.so">
      <![CDATA[
```

```
        <DiameterPeer
          FQDN="%sgw_diameter_fqdn"
          Realm="%base_realm"
          Vendor_Id="45569"
          Product_Name="OpenEPC-7c"
          AcceptUnknownPeers="1"
          DropUnknownOnDisconnect="1"
          Tc="30"
          Workers="1"
          QueueLength="32"
          TransactionTimeout="5"
          SessionsHashSize="128">

          <Acceptor bind="%sgw_mgmt_ipv4" port="%
              sgw_mgmt_diameter_port"/>

<!--         <Peer FQDN="%pcrf_diameter_fqdn" Realm="%base_realm"
    port="%pcrf_mgmt_diameter_port"/> -->

          <Auth id="16777266" vendor="10415" />   <!-- 3GPP Gxx -->
          <Auth id="16777266" />                  <!-- 3GPP Gxx -->

          <Realm name="%base_realm">
            <Route FQDN="%pcrf_diameter_fqdn" metric="10"/>
          </Realm>

          <DefaultRoute FQDN="%pcrf_diameter_fqdn" metric="10"/>
        </DiameterPeer>
      ]]>
    </Module>

</Wharf>
```

MME

```
<?xml version="1.0" encoding="UTF-8"?>
<!--

  OpenEPC 7 - MME Configuration File

  For full documentation on the parameter please check:
    - wharf/cfg/wharf.xml for the core configuration
    - wharf/modules/module_name/module_name.xml for the
       configuration of each individual modules

-->

<Wharf path="%OpenEPC_path_wharf">

  <Core>
    <Debug log="3" memory="6" memory_status="1"/>
    <SharedMemory kbytes="16384" />
    <ProcessMemory kbytes="4096" />
    <WorkerPool count="2" queue_size="8" />
  </Core>

  <Module binaryFile="modules/console/console.so" >
    <![CDATA[
      <WharfConsole>
        <Prompt text=" MME Console >"/>
        <Acceptor type="udp" port="%mme_mgmt_console_port" bind="%
            mme_mgmt_ipv4" />
        <Acceptor type="tcp" port="%mme_mgmt_console_port" bind="%
            mme_mgmt_ipv4" />
      </WharfConsole>
    ]]>
  </Module>

  <Module binaryFile="modules/cdp2/cdp2.so">
    <![CDATA[
      <DiameterPeer
        FQDN="%mme_diameter_fqdn"
        Realm="%base_realm"
        Vendor_Id="45569"
        Product_Name="OpenEPC-7c"
        AcceptUnknownPeers="1"
        DropUnknownOnDisconnect="1"
        Tc="30"
        Workers="4"
        QueueLength="32"
        TransactionTimeout="5"
        SessionsHashSize="128">

```

```
46          <Peer FQDN="%dra_diameter_fqdn"          Realm="%base_realm"
                port="%dra_mgmt_diameter_port"/>
47
48          <Acceptor bind="%mme_mgmt_ipv4" port="%
               mme_mgmt_diameter_port" />
49
50          <Auth id="16777251" vendor="10415"/> <!-- 3GPP S6a/S6d -->
51          <Auth id="16777313" vendor="10415"/> <!-- 3GPP SGd -->
52
53          <Realm name="%base_realm">
54            <Route FQDN="%dra_diameter_fqdn" metric="10"/>
55          </Realm>
56          <DefaultRoute FQDN="%dra_diameter_fqdn" metric="10"/>
57        </DiameterPeer>
58      ]]>
59    </Module>
60
61    <Module binaryFile="modules/mysql/mysql.so"/>
62
63    <Module binaryFile="modules/gsm_l3/gsm_l3.so"/>
64
65    <Module binaryFile="modules/client_s6ad/client_s6ad.so"/>
66
67    <Module binaryFile="modules/sms_codec/sms_codec.so"/>
68
69    <Module binaryFile="modules/addressing/addressing.so" >
70      <![CDATA[
71        <WharfAddressingWRR type="SNAPTR" timeout="300"/>
72      ]]>
73    </Module>
74
75    <Module binaryFile="modules/sctp/sctp.so"/>
76
77    <Module binaryFile="modules/s1ap/s1ap.so" />
78
79    <Module binaryFile="modules/nas/nas.so"/>
80
81    <Module binaryFile="modules/mme/mme.so">
82      <![CDATA[
83        <WharfMME
84          s11_mme="%mme_net_d_ipv4"
85          hash_size="24"
86          relative_capacity="255"
87          cs_not_preferred="no"
88          dst="0"
89          time_zone="4"
90          supported_ta_list="no"
91          enodeb_whitelist="no"
92          sms_store="no">
93          <GUMMEI
94            mcc="%mcc"
```

```
            mnc="%mnc"
            mmegi="%mme_group_id"
            mmec="%mme_code"
            mme_name="%mme_name" />
          <S1-MME address="%mme_net_d_ipv4" />
          <Security
            eea_primary="0"
            eia_primary="1"
            eea_secondary="0"
            eia_secondary="1"/>
          <Database
            hostname="%mme_db_host"
            database="%mme_db_name"
            username="%mme_db_user"
            password="%mme_db_pass"/>
          <Timers
            gtp = "3"
            tc1n = "3"
            sms_validity = "259200" />
          <ESM
            default_dns_ipv4_primary="%dns_net_a_ipv4"
            default_dns_ipv4_secondary="8.8.8.8"
            default_dns_ipv6_primary="%dns_net_a_ipv6"
            default_dns_ipv6_secondary="2001:4860:4860::8888" />
          <EMM attachment_type="2" />
        </WharfMME>
      ]]>
    </Module>

    <Module binaryFile="modules/gtp/gtp.so">
      <![CDATA[
        <GTP>
          <Acceptor id="MME-S11-GTP-C" type="udp" bind="%
              mme_net_d_ipv4" port="%mme_net_d_gtpc_port" />
        </GTP>
      ]]>
    </Module>

  </Wharf>
```

ENODEB2

```
<?xml version="1.0" encoding="UTF-8"?>
<!--

  OpenEPC 7 - eNodeB-L3 #2 Configuration File

  For full documentation on the parameter please check:
    - wharf/cfg/wharf.xml for the core configuration
    - wharf/modules/module_name/module_name.xml for the
      configuration of each individual modules

-->

<Wharf path="%OpenEPC_path_wharf">

  <Core>
    <Debug log="3" memory="6" memory_status="1"/>
    <SharedMemory kbytes="16384" />
    <ProcessMemory kbytes="4096" />
    <WorkerPool count="2" queue_size="8" />
  </Core>

  <Module binaryFile="modules/console/console.so" >
    <![CDATA[
      <WharfConsole>
        <Prompt text=" eNodeB-L3 2 Console >"/>
        <Acceptor type="udp" port="%enodeb2_mgmt_console_port" bind
           ="%enodeb2_mgmt_ipv4" />
        <Acceptor type="tcp" port="%enodeb2_mgmt_console_port" bind
           ="%enodeb2_mgmt_ipv4" />
      </WharfConsole>
    ]]>
  </Module>
  <Module binaryFile="modules/ehcp_messaging/ehcp_messaging.so" />
  <Module binaryFile="modules/ehcp_daemon/ehcp_daemon.so">
    <![CDATA[
      <WharfEHCP_Daemon
        device="net_c"
        domain="%base_realm">
        <DHCP
          local="%enodeb2_net_c_ipv4"
          gateway="192.168.33.30"
          netmask="32"
          dns="%dns_net_a_ipv4" />
        <EHCP
          local="%enodeb2_net_c_ipv6"
          gateway="%enodeb2_net_c_ipv6"
          netmask=""
          dns="%dns_net_a_ipv6" />
```

```
        </WharfEHCP_Daemon>
      ]]>
    </Module>

    <Module binaryFile="modules/gtp/gtp.so">
      <![CDATA[
        <GTP>
          <Acceptor id="GTP-U" type="udp" port="%
              enodeb2_net_d_gtpu_port" bind="%enodeb2_net_d_ipv4" />
        </GTP>
      ]]>
    </Module>

    <Module binaryFile="modules/mysql/mysql.so"/>

    <Module binaryFile="modules/addressing/addressing.so" >
      <![CDATA[
          <WharfAddressingWRR type="SNAPTR" timeout="300">
              <Block domain="mmec%mmec_hex.mmegi%mmegi_hex.mme.%
                  base_realm" service="x-3gpp-mme:x-s1-mme" type="WRR
                  ">
                   <Address ip="%mme_net_d_ipv4" weight="1" />
              </Block>
              <Block domain="rnc%nodeb_rnc_id_hex.rnc.%base_realm"
                  service="x-3gpp-rnc:x-s12" type="WRR">
                   <Address ip="%nodeb_net_d_ipv4" weight="1" />
              </Block>
          </WharfAddressingWRR>
      ]]>
    </Module>

    <Module binaryFile="modules/sctp/sctp.so"/>

    <Module binaryFile="modules/S1AP/s1ap.so">
      <![CDATA[
        <WharfS1AP>
          <Local addr="%enodeb2_net_d_ipv4" port="%
              enodeb2_net_d_s1ap_port" />
          <Remote port="%mme_net_d_s1ap_port" />
        </WharfS1AP>
      ]]>
    </Module>

    <Module binaryFile="modules/gsm_l3/gsm_l3.so"/>

    <Module binaryFile="modules/nas/nas.so"/>

    <Module binaryFile="modules/routing_gtpu/routing_gtpu.so" />

    <Module binaryFile="modules/routing_raw/routing_raw.so" />

```

```
92      <Module binaryFile="modules/routing/routing.so" >
93        <![CDATA[
94          <WharfROUTING>
95            <Extension
96              id="1"
97              dst_table="teid"
98              mod_name="routing_gtpu"
99              ipv4="%enodeb2_net_d_ipv4" />
100           <Extension
101             id="0"
102             src_table="ip"
103             mod_name="routing_raw"
104             interface="net_c"
105             ipv4="%enodeb2_net_c_ipv4"
106             ipv6="%enodeb2_net_c_ipv6"/>
107         </WharfROUTING>
108       ]]>
109     </Module>
110
111     <Module binaryFile="modules/enodeb/enodeb.so" >
112       <![CDATA[
113         <WharfEnodeb
114           s1_u="%enodeb2_net_d_ipv4"
115           mnc="%mnc"
116           mcc="%mcc"
117           tac="x%enodeb2_tac_hb%enodeb2_tac_lb"
118           cellid="4356"
119           an_interface="net_c"
120           mme_code="%mme_code"
121           mme_group_id="%mme_group_id"
122           default_imsi="001011234567890"
123           default_apn="%default_apn_short">
124         </WharfEnodeb>
125       ]]>
126     </Module>
127
128     <Module binaryFile="modules/X2AP/x2ap.so">
129       <![CDATA[
130         <x2ap>
131           <sctp_ep local_ip="%enodeb2_net_d_ipv4" local_port="%
                enodeb2_net_d_x2ap_port"/>
132           <sctp_association id="1" remote_ip="%enodeb_net_d_ipv4"
                remote_port="%enodeb_net_d_x2ap_port"/>
133         </x2ap>
134       ]]>
135     </Module>
136
137   </Wharf>
```

CLIENT_NAME

```
<?xml version="1.0" encoding="UTF-8"?>
<!--

 OpenEPC 7 - MM-Name Configuration File

 For full documentation on the parameter please check:
        - wharf/cfg/wharf.xml for the core configuration
        - wharf/modules/module_name/module_name.xml for the
           configuration of each individual modules

-->

<Wharf path="%OpenEPC_path_wharf">

 <Core>
        <Debug log="3" memory="6" memory_status="1"/>
        <SharedMemory kbytes="16384" />
        <ProcessMemory kbytes="4096" />
        <WorkerPool count="2" queue_size="8" />
 </Core>

 <Module binaryFile="modules/console/console.so" >
        <![CDATA[
                <WharfConsole>
                        <Prompt text="MM-Name Console>"/>
                        <Acceptor type="udp" port="%
                           mm_lo_console_port" bind="127.0.0.1" />
                </WharfConsole>
        ]]>
 </Module>

 <Module binaryFile="modules/mm_eth/mm_eth.so" >
        <![CDATA[
                <MM_ETH link_check_interval="3" >
                        <Interface id="LTE"         name="an_lte"
                           />
                </MM_ETH>
        ]]>
 </Module>

 <Module binaryFile="modules/mm_ip/mm_ip.so" >
        <![CDATA[
                <MM_IP>
                        <EHCP
                                delay="5000"
                                retries="1" />
                        <DHCP
                                delay="5000"
```

```
                                        retries="1" />
                        </MM_IP>
                ]]>
        </Module>

        <Module binaryFile="modules/ehcp_messaging/ehcp_messaging.so" />

        <Module binaryFile="modules/mm_gui_comm/mm_gui_comm.so" />

        <Module binaryFile="modules/mm/mm.so" >
                <![CDATA[
                        <MM>
                                <NetworkList
                                        file="%OpenEPC_path_etc/mm_network.
                                            xml"
                                        refresh_interval="10"/>
                                <Client imsi="%imsi_name"/>
                        </MM>
                ]]>
        </Module>

        <Module binaryFile="modules/syncml/syncml.so"/>

        <Module binaryFile="modules/httpc/httpc.so"/>

</Wharf>
```

ANEXO ORQUESTACIÓN (HEAT) TEMPLATES

OPERADOR VERDE

```
        heat_template_version: 2014-10-16

description: BISDN GmbH, 2017. Requires existing public network
    named 'public' Template to deploy GiLAN with dummy networks.
    Should work with no prior networks/routers configured.

parameters: #this section is if you want to give the parameters when
     you are creating the stack
 image_client_bob:
  type: string
  label: Image name or ID
  description: Image to be used for compute instance
  default: client-bob #CirrOS 0.3.5 DT uses "CentOS 7", img
      available in our openstack
 image_enodeb:
  type: string
  label: Image name or ID
  description: Image to be used for compute instance
  default: enodeb #CirrOS 0.3.5 DT uses "CentOS 7", img available in
      our openstack
 image_pgw:
  type: string
  label: Image name or ID
  description: Image to be used for compute instance
  default: pgw #CirrOS 0.3.5 DT uses "CentOS 7", img available in
      our openstack
 image_sgw:
  type: string
  label: Image name or ID
  description: Image to be used for compute instance
  default: sgw #CirrOS 0.3.5 DT uses "CentOS 7", img available in
      our openstack
 image_epc-enablers:
  type: string
  label: Image name or ID
  description: Image to be used for compute instance
  default: epc-enablers #CirrOS 0.3.5 DT uses "CentOS 7", img
      available in our openstack
 image_mme:
  type: string
  label: Image name or ID
  description: Image to be used for compute instance
```

```
    default: mme #CirrOS 0.3.5 DT uses "CentOS 7", img available in our openstack
  flavor: #the flavor used
    type: string
    label: flavor
    description: flavor to be used
    default: openepc

  keys:
    type: string
    description: the SSH keys to be installed on the servers
    default: GiLAN
  dns_nameserver:
     type: string
     description: the IP address of YOUR local/global dns nameserver.
     default: "8.8.8.8"
  public_network: #the public network to be used
    type: string
    label: Public network name or ID
    description: Public network
    default: provider # Name of public network, needs to be external and already exist in OpenStack


resources: #in this section, the resources are defined

# NETWORKS
  heat_net_mgmt:
    type: OS::Neutron::Net
    properties:
      name: HEAT_NET_MNGT

  heat_net_an_lte:
    type: OS::Neutron::Net
    properties:
      name: HEAT_AN_LTE

  heat_net_a:
    type: OS::Neutron::Net
    properties:
      name: HEAT_NET_A

  heat_net_gw:
    type: OS::Neutron::Net
    properties:
      name: HEAT_NET_GW


  heat_net_d:
    type: OS::Neutron::Net
    properties:
```

```
    name: HEAT_NET_D

# SUBNETS
 heat_net_mgmt_subnet: #creating the subnets
  type: OS::Neutron::Subnet
  depends_on: heat_net_mgmt
  properties:
   network_id: { get_resource: heat_net_mgmt }
   cidr: "192.168.254.0/24"
   gateway_ip: "192.168.254.2"
   dns_nameservers:
     - { get_param: dns_nameserver }

 heat_net_an_lte_subnet: #creating the subnets
  type: OS::Neutron::Subnet
  depends_on: heat_net_an_lte
  properties:
   network_id: { get_resource: heat_net_an_lte }
   cidr: "192.168.3.0/24"

 heat_net_a_subnet:
  type: OS::Neutron::Subnet
  depends_on: heat_net_a
  properties:
   network_id: { get_resource: heat_net_a }
   cidr: "192.168.1.0/24"

 heat_net_d_subnet:
  type: OS::Neutron::Subnet
  depends_on: heat_net_d
  properties:
   network_id: { get_resource: heat_net_d }
   cidr: "192.168.4.0/24"

 heat_net_gw_subnet:
  type: OS::Neutron::Subnet
  depends_on: heat_net_gw
  properties:
   network_id: { get_resource: heat_net_gw }
   cidr: "192.168.3.0/24"
   gateway_ip: "192.168.3.1"


# ROUTERS
 heat_net_gw_router:
   type: OS::Neutron::Router
   depends_on: heat_net_gw_subnet
   properties:
     name: heat_net_gw_router
     external_gateway_info:
       network: { get_param: public_network }
```

```
135
136   heat_net_mgmt_router:
137    type: OS::Neutron::Router
138    depends_on: heat_net_mgmt_subnet
139    properties:
140      name: heat_net_mgmt_router
141      external_gateway_info:
142        network: { get_param: public_network }
143
144
145   # INTERFACES
146    public-interface-gw: #creating the router interface, on an existing
           router
147    type: OS::Neutron::RouterInterface
148    depends_on: heat_net_gw_router
149    properties:
150     router_id: { get_resource: heat_net_gw_router }
151     subnet: { get_resource: heat_net_gw_subnet }
152
153    public-interface: #creating the router interface, on an existing
          router
154    type: OS::Neutron::RouterInterface
155    depends_on: heat_net_mgmt_router
156    properties:
157     router_id: { get_resource: heat_net_mgmt_router } #this is the id
            of the router used *it can be defined in the parameters if
           we want
158     subnet: { get_resource: heat_net_mgmt_subnet } #the subnet the
           interface is created for
159
160   # PORTS for MAC-addresses
161   #-- Client_bob
162    heat_client_bob_net_mgmt_port:
163     type: OS::Neutron::Port
164     depends_on: heat_net_mgmt_subnet
165     properties:
166       network_id: { get_resource: heat_net_mgmt }
167       mac_address: 52:54:00:8d:98:b8
168       port_security_enabled: false
169       fixed_ips:
170         - ip_address: 192.168.254.101
171
172    heat_client_bob_net_an_lte_port:
173     type: OS::Neutron::Port
174     depends_on: heat_net_an_lte_subnet
175     properties:
176       network_id: { get_resource: heat_net_an_lte }
177       mac_address: 52:54:00:89:9f:52
178       port_security_enabled: false
179       fixed_ips:
180         - ip_address: 192.168.3.101
```

```
#--Enodeb
 heat_enodeb_net_mgmt_port:
  type: OS::Neutron::Port
  depends_on: heat_net_mgmt_subnet
  properties:
    network_id: { get_resource: heat_net_mgmt }
    mac_address: 52:54:00:1b:14:7e
    port_security_enabled: false
    fixed_ips:
      - ip_address: 192.168.254.91

 heat_enodeb_net_an_lte_port:
  type: OS::Neutron::Port
  depends_on: heat_net_an_lte_subnet
  properties:
    network_id: { get_resource: heat_net_an_lte }
    mac_address: 52:54:00:e1:c4:fb
    port_security_enabled: false
    fixed_ips:
      - ip_address: 192.168.3.30
#      - ip_address: 192.168.33.30

 heat_enodeb_net_d_port:
  type: OS::Neutron::Port
  depends_on: heat_net_d_subnet
  properties:
    network_id: { get_resource: heat_net_d }
    mac_address: 52:54:00:e3:23:a9
    port_security_enabled: false
    fixed_ips:
      - ip_address: 192.168.4.91
#--SGW
 heat_sgw_net_mgmt_port:
  type: OS::Neutron::Port
  depends_on: heat_net_mgmt_subnet
  properties:
    network_id: { get_resource: heat_net_mgmt }
    mac_address: 52:54:00:77:2f:b3
    port_security_enabled: false
    fixed_ips:
      - ip_address: 192.168.254.20

 heat_sgw_net_b_port:
  type: OS::Neutron::Port
  properties:
    network_id: { get_param: public_network }
    mac_address: 52:54:00:b8:57:31
    port_security_enabled: false
    fixed_ips:
      - ip_address: 10.2.36.20

```

```
 heat_sgw_net_d_port:
  type: OS::Neutron::Port
  depends_on: heat_net_d_subnet
  properties:
    network_id: { get_resource: heat_net_d }
    mac_address: 52:54:00:91:ef:91
    port_security_enabled: false
    fixed_ips:
      - ip_address: 192.168.4.20
#--PGW
 heat_pgw_net_mgmt_port:
  type: OS::Neutron::Port
  depends_on: heat_net_mgmt_subnet
  properties:
    network_id: { get_resource: heat_net_mgmt }
    mac_address: 52:54:00:1b:0a:d7
    port_security_enabled: false
    fixed_ips:
      - ip_address: 192.168.254.11

 heat_pgw_net_b_port:
  type: OS::Neutron::Port
  properties:
    network_id: { get_param: public_network }
    mac_address: 52:54:00:9d:da:8a
    port_security_enabled: false
    fixed_ips:
      - ip_address: 10.2.36.11

 heat_pgw_net_a_port:
  type: OS::Neutron::Port
  depends_on: heat_net_a_subnet
  properties:
    network_id: { get_resource: heat_net_a }
    mac_address: 52:54:00:56:1d:1d
    port_security_enabled: false
    fixed_ips:
      - ip_address: 192.168.1.11

 heat_pgw_net_gw_port:
   type: OS::Neutron::Port
   depends_on: heat_net_gw_subnet
   properties:
     network_id: { get_resource: heat_net_gw }
     mac_address: 52:54:00:75:68:d8
     port_security_enabled: false
     fixed_ips:
       - ip_address: 192.168.3.11
       - ip_address: 192.168.3.101

#--Epc-enablers
```

```
  heat_epc-enablers_net_mgmt_port:
   type: OS::Neutron::Port
   depends_on: heat_net_mgmt_subnet
   properties:
     network_id: { get_resource: heat_net_mgmt }
     mac_address: 52:54:00:38:d5:16
     port_security_enabled: false
     fixed_ips:
#       - ip_address: 192.168.254.30 #default
#       - ip_address: 192.168.254.32 #dashboard
#       - ip_address: 192.168.254.33 #hss
#       - ip_address: 192.168.254.40 #dns
#       - ip_address: 192.168.254.45
#       - ip_address: 192.168.254.46
#       - ip_address: 192.168.254.47
#       - ip_address: 192.168.254.70
       - ip_address: 192.168.254.170 #dra_ip

  floating_ip_epc:
     type: OS::Neutron::FloatingIP
     properties:
       floating_network: {get_param: public_network}
       port_id: { get_resource: heat_epc-enablers_net_mgmt_port }
       floating_ip_address: 10.2.35.170


  heat_epc-enablers_net_a_port:
   type: OS::Neutron::Port
   depends_on: heat_net_a_subnet
   properties:
     network_id: { get_resource: heat_net_a }
     mac_address: 52:54:00:9b:f9:e6
     port_security_enabled: false
     fixed_ips:
       - ip_address: 192.168.1.32
#       - ip_address: 192.168.1.33
#       - ip_address: 192.168.1.40
#       - ip_address: 192.168.1.45
#       - ip_address: 192.168.1.46
#       - ip_address: 192.168.1.47
#       - ip_address: 192.168.1.70
#       - ip_address: 192.168.1.170
#
#

  #--MME
   heat_mme_net_mgmt_port:
    type: OS::Neutron::Port
    depends_on: heat_net_mgmt_subnet
    properties:
      network_id: { get_resource: heat_net_mgmt }
```

```
334       mac_address: 52:54:00:62:79:96
335       port_security_enabled: false
336       fixed_ips:
337         - ip_address: 192.168.254.80
338
339    heat_mme_net_d_port:
340     type: OS::Neutron::Port
341     depends_on: heat_net_d_subnet
342     properties:
343       network_id: { get_resource: heat_net_d }
344       mac_address: 52:54:00:68:9e:ad
345       port_security_enabled: false
346       fixed_ips:
347         - ip_address: 192.168.4.80
348
349
350    # SERVERS/INSTANCES
351    heat_client_bob:
352     type: OS::Nova::Server
353     depends_on: [heat_client_bob_net_mgmt_port,
           heat_client_bob_net_an_lte_port, heat_enodeb ]
354     properties:
355      image: { get_param: image_client_bob }
356      name: INST_CLIENT_BOB
357      flavor: { get_param: flavor }
358      key_name: { get_param: keys }
359      networks:
360       - port: {get_resource: heat_client_bob_net_mgmt_port}
361       - port: {get_resource: heat_client_bob_net_an_lte_port}
362
363    heat_enodeb:
364     type: OS::Nova::Server
365     depends_on: [ heat_enodeb_net_mgmt_port,
           heat_enodeb_net_an_lte_port, heat_enodeb_net_d_port, heat_sgw
           ]
366     properties:
367      image: { get_param: image_enodeb }
368      name: INST_ENODEB
369      flavor: { get_param: flavor }
370      key_name: { get_param: keys }
371      networks:
372        - port: {get_resource: heat_enodeb_net_mgmt_port }
373        - port: {get_resource: heat_enodeb_net_an_lte_port }
374        - port: {get_resource: heat_enodeb_net_d_port }
375
376    heat_sgw:
377     type: OS::Nova::Server
378     depends_on: [ heat_sgw_net_mgmt_port, heat_sgw_net_b_port,
           heat_sgw_net_d_port, heat_epc-enablers, heat_mme, heat_pgw ]
379     properties:
380      image: { get_param: image_sgw }
```

```
    name: INST_SGW
    flavor: { get_param: flavor }
    key_name: { get_param: keys }
    networks:
     - port: {get_resource: heat_sgw_net_mgmt_port }
     - port: {get_resource: heat_sgw_net_b_port }
     - port: {get_resource: heat_sgw_net_d_port }

  heat_pgw:
   type: OS::Nova::Server
   depends_on: [ heat_pgw_net_mgmt_port, heat_pgw_net_a_port,
      heat_pgw_net_b_port,heat_pgw_net_gw_port, heat_epc-enablers,
      heat_mme ]
   properties:
    image: { get_param: image_pgw }
    name: INST_PGW
    flavor: { get_param: flavor }
    key_name: { get_param: keys }
    networks:
     - port: {get_resource: heat_pgw_net_mgmt_port }
     - port: {get_resource: heat_pgw_net_a_port }
     - port: {get_resource: heat_pgw_net_b_port }
     - port: {get_resource: heat_pgw_net_gw_port }


  heat_mme:
   type: OS::Nova::Server
   depends_on: [ heat_mme_net_mgmt_port, heat_mme_net_d_port,
      heat_epc-enablers ]
   properties:
    image: { get_param: image_mme }
    name: INST_MME
    flavor: { get_param: flavor }
    key_name: { get_param: keys }
    networks:
     - port: {get_resource: heat_mme_net_mgmt_port }
     - port: {get_resource: heat_mme_net_d_port }

  heat_epc-enablers:
   type: OS::Nova::Server
   depends_on: [ heat_epc-enablers_net_mgmt_port, heat_epc-
      enablers_net_a_port ]
   properties:
    image: { get_param: image_epc-enablers }
    name: INST_EPC-ENABLERS
    flavor: { get_param: flavor }
    key_name: { get_param: keys }
    networks:
     - port: {get_resource: heat_epc-enablers_net_mgmt_port }
     - port: {get_resource: heat_epc-enablers_net_a_port }
```

OPERADOR ROJO

```
heat_template_version: 2014-10-16

description: BISDN GmbH, 2017. Requires existing public network
    named 'public' Template to deploy GiLAN with dummy networks.
    Should work with no prior networks/routers configured.

parameters: #this section is if you want to give the parameters when
     you are creating the stack
 image_client_bob:
  type: string
  label: Image name or ID
  description: Image to be used for compute instance
  default: client-bob_2 #CirrOS 0.3.5 DT uses "CentOS 7", img
      available in our openstack
 image_enodeb:
  type: string
  label: Image name or ID
  description: Image to be used for compute instance
  default: enodeb_2 #CirrOS 0.3.5 DT uses "CentOS 7", img available
      in our openstack
 image_pgw:
  type: string
  label: Image name or ID
  description: Image to be used for compute instance
  default: pgw_2 #CirrOS 0.3.5 DT uses "CentOS 7", img available in
      our openstack
 image_sgw:
  type: string
  label: Image name or ID
  description: Image to be used for compute instance
  default: sgw_2 #CirrOS 0.3.5 DT uses "CentOS 7", img available in
      our openstack
 image_epc-enablers:
  type: string
  label: Image name or ID
  description: Image to be used for compute instance
  default: epc-enablers_2 #CirrOS 0.3.5 DT uses "CentOS 7", img
      available in our openstack
 image_mme:
  type: string
  label: Image name or ID
  description: Image to be used for compute instance
  default: mme_2 #CirrOS 0.3.5 DT uses "CentOS 7", img available in
      our openstack
 flavor: #the flavor used
  type: string
  label: flavor
  description: flavor to be used
```

```
    default: openepc
  keys:
    type: string
    description: the SSH keys to be installed on the servers
    default: GiLAN
  dns_nameserver:
     type: string
     description: the IP address of YOUR local/global dns nameserver.
     default: "8.8.8.8"
  public_network: #the public network to be used
    type: string
    label: Public network name or ID
    description: Public network
    default: provider # Name of public network

resources: #in this section, the resources are defined

# NETWORKS
  heat_net_mgmt:
    type: OS::Neutron::Net
    properties:
      name: HEAT_NET_MNGT_2

  heat_net_an_lte:
    type: OS::Neutron::Net
    properties:
      name: HEAT_AN_LTE_2

  heat_net_a:
    type: OS::Neutron::Net
    properties:
      name: HEAT_NET_A_2

  heat_net_gw:
    type: OS::Neutron::Net
    properties:
      name: HEAT_NET_GW_2

  heat_net_d:
    type: OS::Neutron::Net
    properties:
      name: HEAT_NET_D_2

# SUBNETS
  heat_net_mgmt_subnet: #creating the subnets
    type: OS::Neutron::Subnet
    depends_on: heat_net_mgmt
    properties:
      network_id: { get_resource: heat_net_mgmt }
      cidr: "192.168.253.0/24"
      dns_nameservers:
```

```
      - { get_param: dns_nameserver }

  heat_net_an_lte_subnet: #creating the subnets
   type: OS::Neutron::Subnet
   depends_on: heat_net_an_lte
   properties:
    network_id: { get_resource: heat_net_an_lte }
    cidr: "192.168.33.0/24"

  heat_net_a_subnet:
   type: OS::Neutron::Subnet
   depends_on: heat_net_a
   properties:
    network_id: { get_resource: heat_net_a }
    cidr: "192.168.11.0/24"

  heat_net_d_subnet:
   type: OS::Neutron::Subnet
   depends_on: heat_net_d
   properties:
    network_id: { get_resource: heat_net_d }
    cidr: "192.168.14.0/24"

  heat_net_gw_subnet:
   type: OS::Neutron::Subnet
   depends_on: heat_net_gw
   properties:
    network_id: { get_resource: heat_net_gw }
    cidr: "192.168.33.0/24"


 # ROUTERS
  heat_net_mgmt_router:
   type: OS::Neutron::Router
   depends_on: heat_net_mgmt_subnet
   properties:
     name: heat_net_mgmt_router
     external_gateway_info:
       network: { get_param: public_network }

  heat_net_gw_router:
   type: OS::Neutron::Router
   depends_on: heat_net_gw_subnet
   properties:
     name: heat_net_gw_router
     external_gateway_info:
       network: { get_param: public_network }



 # INTERFACES
```

```
142   public-interface: #creating the router interface, on an existing
          router
143     type: OS::Neutron::RouterInterface
144     depends_on: heat_net_mgmt_router
145     properties:
146       router_id: { get_resource: heat_net_mgmt_router } #this is the id
              of the router used *it can be defined in the parameters if
            we want
147       subnet: { get_resource: heat_net_mgmt_subnet } #the subnet the
            interface is created for
148
149   public-interface_gw: #creating the router interface, on an existing
          router
150     type: OS::Neutron::RouterInterface
151     depends_on: heat_net_gw_router
152     properties:
153       router_id: { get_resource: heat_net_gw_router }
154       subnet: { get_resource: heat_net_gw_subnet }
155
156   # PORTS for MAC-addresses
157   #-- Client_bob
158    heat_client_bob_net_mgmt_port:
159     type: OS::Neutron::Port
160     depends_on: heat_net_mgmt_subnet
161     properties:
162       network_id: { get_resource: heat_net_mgmt }
163       mac_address: 52:54:00:b0:11:9f
164       port_security_enabled: false
165       fixed_ips:
166         - ip_address: 192.168.253.101
167
168    heat_client_bob_net_an_lte_port:
169     type: OS::Neutron::Port
170     depends_on: heat_net_an_lte_subnet
171     properties:
172       network_id: { get_resource: heat_net_an_lte }
173       mac_address: 52:54:00:d3:c8:fe
174       port_security_enabled: false
175       fixed_ips:
176         - ip_address: 192.168.33.101
177   #--Enodeb
178    heat_enodeb_net_mgmt_port:
179     type: OS::Neutron::Port
180     depends_on: heat_net_mgmt_subnet
181     properties:
182       network_id: { get_resource: heat_net_mgmt }
183       mac_address: 52:54:00:fb:6b:f5
184       port_security_enabled: false
185       fixed_ips:
186         - ip_address: 192.168.253.91
187
```

```
  heat_enodeb_net_an_lte_port:
   type: OS::Neutron::Port
   depends_on: heat_net_an_lte_subnet
   properties:
     network_id: { get_resource: heat_net_an_lte }
     mac_address: 52:54:00:45:29:eb
     port_security_enabled: false
     fixed_ips:
       - ip_address: 192.168.33.30
#       - ip_address: 192.168.33.30

  heat_enodeb_net_d_port:
   type: OS::Neutron::Port
   depends_on: heat_net_d_subnet
   properties:
     network_id: { get_resource: heat_net_d }
     mac_address: 52:54:00:18:7b:c4
     port_security_enabled: false
     fixed_ips:
       - ip_address: 192.168.14.91
#--SGW
  heat_sgw_net_mgmt_port:
   type: OS::Neutron::Port
   depends_on: heat_net_mgmt_subnet
   properties:
     network_id: { get_resource: heat_net_mgmt }
     mac_address: 52:54:00:bd:b9:3f
     port_security_enabled: false
     fixed_ips:
       - ip_address: 192.168.253.20

  heat_sgw_net_b_port:
   type: OS::Neutron::Port
   properties:
     network_id: { get_param: public_network }
     mac_address: 52:54:00:d9:3f:e2
     port_security_enabled: false
     fixed_ips:
       - ip_address: 10.34.36.20

  heat_sgw_net_d_port:
   type: OS::Neutron::Port
   depends_on: heat_net_d_subnet
   properties:
     network_id: { get_resource: heat_net_d }
     mac_address: 52:54:00:de:7c:d6
     port_security_enabled: false
     fixed_ips:
       - ip_address: 192.168.14.20
#--PGW
  heat_pgw_net_mgmt_port:
```

```
    type: OS::Neutron::Port
    depends_on: heat_net_mgmt_subnet
    properties:
      network_id: { get_resource: heat_net_mgmt }
      mac_address: 52:54:00:04:b2:31
      port_security_enabled: false
      fixed_ips:
        - ip_address: 192.168.253.11

  heat_pgw_net_b_port:
    type: OS::Neutron::Port
    properties:
      network_id: { get_param: public_network }
      mac_address: 52:54:00:2d:f0:80
      port_security_enabled: false
      fixed_ips:
        - ip_address: 10.34.36.11

  heat_pgw_net_a_port:
    type: OS::Neutron::Port
    depends_on: heat_net_a_subnet
    properties:
      network_id: { get_resource: heat_net_a }
      mac_address: 52:54:00:f0:9a:43
      port_security_enabled: false
      fixed_ips:
        - ip_address: 192.168.11.11

  heat_pgw_net_gw_port:
     type: OS::Neutron::Port
     depends_on: heat_net_gw_subnet
     properties:
       network_id: { get_resource: heat_net_gw }
       mac_address: 52:54:00:21:b9:c1
       port_security_enabled: false
       fixed_ips:
         - ip_address: 192.168.33.11

 #--Epc-enablers
  heat_epc-enablers_net_mgmt_port:
    type: OS::Neutron::Port
    depends_on: heat_net_mgmt_subnet
    properties:
      network_id: { get_resource: heat_net_mgmt }
      mac_address: 52:54:00:25:e8:81
      port_security_enabled: false
      fixed_ips:
 #       - ip_address: 192.168.253.30 #default
 #       - ip_address: 192.168.253.32 #dashboard
 #       - ip_address: 192.168.253.33 #hss
 #       - ip_address: 192.168.253.40 #dns
```

```
#       - ip_address: 192.168.253.45
#       - ip_address: 192.168.253.46
#       - ip_address: 192.168.253.47
#       - ip_address: 192.168.254.70
        - ip_address: 192.168.253.170 #dra_ip

  floating_ip_epc:
     type: OS::Neutron::FloatingIP
     properties:
       floating_network: {get_param: public_network}
       port_id: { get_resource: heat_epc-enablers_net_mgmt_port }
       floating_ip_address: 10.34.35.170

#  association:
#   type: OS::Nova::FloatingIPAssociation
#   properties:
#     floating_ip: { get_resource: floating_ip_epc }
#     port_id: { get_resource: heat_epc-enablers_net_mgmt_port }

  heat_epc-enablers_net_a_port:
   type: OS::Neutron::Port
   depends_on: heat_net_a_subnet
   properties:
     network_id: { get_resource: heat_net_a }
     mac_address: 52:54:00:c6:d5:84
     port_security_enabled: false
     fixed_ips:
       - ip_address: 192.168.11.32
#       - ip_address: 192.168.11.33
#       - ip_address: 192.168.11.40
#       - ip_address: 192.168.11.45
#       - ip_address: 192.168.11.46
#       - ip_address: 192.168.11.47
#       - ip_address: 192.168.11.70
#       - ip_address: 192.168.11.170
#
#

#--MME
  heat_mme_net_mgmt_port:
   type: OS::Neutron::Port
   depends_on: heat_net_mgmt_subnet
   properties:
     network_id: { get_resource: heat_net_mgmt }
     mac_address: 52:54:00:ce:95:1c
     port_security_enabled: false
     fixed_ips:
       - ip_address: 192.168.253.80

  heat_mme_net_d_port:
   type: OS::Neutron::Port
```

```
341     depends_on: heat_net_d_subnet
342     properties:
343       network_id: { get_resource: heat_net_d }
344       mac_address: 52:54:00:5f:e4:00
345       port_security_enabled: false
346       fixed_ips:
347         - ip_address: 192.168.14.80
348
349   # SERVERS/INSTANCES
350    heat_client_bob:
351     type: OS::Nova::Server
352     depends_on: [heat_client_bob_net_mgmt_port,
            heat_client_bob_net_an_lte_port, heat_enodeb ]
353     properties:
354      image: { get_param: image_client_bob }
355      name: INST_CLIENT_BOB_2
356      flavor: { get_param: flavor }
357      key_name: { get_param: keys }
358      networks:
359       - port: {get_resource: heat_client_bob_net_mgmt_port}
360       - port: {get_resource: heat_client_bob_net_an_lte_port}
361
362    heat_enodeb:
363     type: OS::Nova::Server
364     depends_on: [ heat_enodeb_net_mgmt_port,
            heat_enodeb_net_an_lte_port, heat_enodeb_net_d_port, heat_sgw
            ]
365     properties:
366      image: { get_param: image_enodeb }
367      name: INST_ENODEB_2
368      flavor: { get_param: flavor }
369      key_name: { get_param: keys }
370      networks:
371        - port: {get_resource: heat_enodeb_net_mgmt_port }
372        - port: {get_resource: heat_enodeb_net_an_lte_port }
373        - port: {get_resource: heat_enodeb_net_d_port }
374
375    heat_sgw:
376     type: OS::Nova::Server
377     depends_on: [ heat_sgw_net_mgmt_port, heat_sgw_net_b_port,
            heat_sgw_net_d_port, heat_epc-enablers, heat_mme, heat_pgw ]
378     properties:
379      image: { get_param: image_sgw }
380      name: INST_SGW_2
381      flavor: { get_param: flavor }
382      key_name: { get_param: keys }
383      networks:
384       - port: {get_resource: heat_sgw_net_mgmt_port }
385       - port: {get_resource: heat_sgw_net_b_port }
386       - port: {get_resource: heat_sgw_net_d_port }
387
```

```
388   heat_pgw:
389    type: OS::Nova::Server
390    depends_on: [ heat_pgw_net_mgmt_port, heat_pgw_net_a_port,
          heat_pgw_net_b_port,heat_pgw_net_gw_port, heat_epc-enablers,
          heat_mme ]
391    properties:
392     image: { get_param: image_pgw }
393     name: INST_PGW_2
394     flavor: { get_param: flavor }
395     key_name: { get_param: keys }
396     networks:
397      - port: {get_resource: heat_pgw_net_mgmt_port }
398      - port: {get_resource: heat_pgw_net_a_port }
399      - port: {get_resource: heat_pgw_net_b_port }
400      - port: {get_resource: heat_pgw_net_gw_port }
401
402
403   heat_mme:
404    type: OS::Nova::Server
405    depends_on: [ heat_mme_net_mgmt_port, heat_mme_net_d_port,
          heat_epc-enablers ]
406    properties:
407     image: { get_param: image_mme }
408     name: INST_MME_2
409     flavor: { get_param: flavor }
410     key_name: { get_param: keys }
411     networks:
412      - port: {get_resource: heat_mme_net_mgmt_port }
413      - port: {get_resource: heat_mme_net_d_port }
414
415   heat_epc-enablers:
416    type: OS::Nova::Server
417    depends_on: [ heat_epc-enablers_net_mgmt_port, heat_epc-
          enablers_net_a_port ]
418    properties:
419     image: { get_param: image_epc-enablers }
420     name: INST_EPC-ENABLERS_2
421     flavor: { get_param: flavor }
422     key_name: { get_param: keys }
423     networks:
424      - port: {get_resource: heat_epc-enablers_net_mgmt_port }
425      - port: {get_resource: heat_epc-enablers_net_a_port }
```

CLIENT_ALICE

```
heat_template_version: 2014-10-16

description: BISDN GmbH, 2017. Requires existing public network
    named 'public' Template to deploy GiLAN clients. Should work
    with an operator already deploy.

parameters: #this section is if you want to give the parameters when
     you are creating the stack
 image_client_alice_2:
  type: string
  label: Image name or ID
  description: Image to be used for compute instance
  default: client-alice_2 #CirrOS 0.3.5 DT uses "CentOS 7", img
      available in our openstack
 flavor: #the flavor used
  type: string
  label: flavor
  description: flavor to be used
  default: openepc
 keys:
  type: string
  description: the SSH keys to be installed on the servers
  default: GiLAN
 dns_nameserver:
   type: string
   description: the IP address of YOUR local/global dns nameserver.
   default: "8.8.8.8"
 heat_net_mgmt:
   type: string
   description: mgmt network
   default: HEAT_NET_MNGT
 heat_net_an_lte:
   type: string
   description: an_lte network
   default: HEAT_AN_LTE

resources: #in this section, the resources are defined
# PORTS for MAC-addresses
# #-- Client_alice_2
 heat_client_alice_2_net_mgmt_port:
  type: OS::Neutron::Port
  properties:
    network_id: { get_param: heat_net_mgmt }
    mac_address: 52:54:00:95:91:09
    port_security_enabled: false
    fixed_ips:
      - ip_address: 192.168.254.100

```

```
 heat_client_alice_2_net_an_lte_port:
  type: OS::Neutron::Port
  properties:
    network_id: { get_param: heat_net_an_lte }
    mac_address: 52:54:00:67:9e:5c
    port_security_enabled: false
    fixed_ips:
      - ip_address: 192.168.3.100
#SERVER
 heat_client_alice_2:
  type: OS::Nova::Server
  depends_on: [heat_client_alice_2_net_mgmt_port,
     heat_client_alice_2_net_an_lte_port ]
  properties:
   image: { get_param: image_client_alice_2 }
   name: INST_ROAMING_OP2_ALICE
   flavor: { get_param: flavor }
   key_name: { get_param: keys }
   networks:
    - port: {get_resource: heat_client_alice_2_net_mgmt_port}
    - port: {get_resource: heat_client_alice_2_net_an_lte_port}
```

CLIENT_CHARLIE

```
heat_template_version: 2014-10-16

description: BISDN GmbH, 2017. Requires existing public network
    named 'public' Template to deploy GiLAN clients. Should work
    with an operator already deploy.

parameters: #this section is if you want to give the parameters when
     you are creating the stack
 image_client_charlie_2:
  type: string
  label: Image name or ID
  description: Image to be used for compute instance
  default: client-charlie_2 #CirrOS 0.3.5 DT uses "CentOS 7", img
      available in our openstack
 flavor: #the flavor used
  type: string
  label: flavor
  description: flavor to be used
  default: openepc
 keys:
  type: string
  description: the SSH keys to be installed on the servers
  default: GiLAN
 dns_nameserver:
   type: string
   description: the IP address of YOUR local/global dns nameserver.
   default: "8.8.8.8"
 heat_net_mgmt:
   type: string
   description: mgmt network
   default: HEAT_NET_MNGT
 heat_net_an_lte:
   type: string
   description: an_lte network
   default: HEAT_AN_LTE

resources: #in this section, the resources are defined
# PORTS for MAC-addresses
# #-- Client_charlie_2
 heat_client_charlie_2_net_mgmt_port:
  type: OS::Neutron::Port
  properties:
    network_id: { get_param: heat_net_mgmt }
    mac_address: 52:54:00:59:a5:09
    port_security_enabled: false
    fixed_ips:
      - ip_address: 192.168.254.102

```

```
 heat_client_charlie_2_net_an_lte_port:
  type: OS::Neutron::Port
  properties:
    network_id: { get_param: heat_net_an_lte }
    mac_address: 52:54:00:76:e9:5c
    port_security_enabled: false
    fixed_ips:
      - ip_address: 192.168.3.102
#SERVER
 heat_client_charlie_2:
  type: OS::Nova::Server
  depends_on: [heat_client_charlie_2_net_mgmt_port,
     heat_client_charlie_2_net_an_lte_port ]
  properties:
   image: { get_param: image_client_charlie_2 }
   name: INST_ROAMING_OP2_CHARLIE
   flavor: { get_param: flavor }
   key_name: { get_param: keys }
   networks:
    - port: {get_resource: heat_client_charlie_2_net_mgmt_port}
    - port: {get_resource: heat_client_charlie_2_net_an_lte_port}
```

CLIENT_DAVE

```
heat_template_version: 2014-10-16

description: BISDN GmbH, 2017. Requires existing public network
    named 'public' Template to deploy GiLAN clients. Should work
    with an operator already deploy.

parameters: #this section is if you want to give the parameters when
     you are creating the stack
 image_client_dave_2:
  type: string
  label: Image name or ID
  description: Image to be used for compute instance
  default: client-dave_2 #CirrOS 0.3.5 DT uses "CentOS 7", img
      available in our openstack
 flavor: #the flavor used
  type: string
  label: flavor
  description: flavor to be used
  default: openepc
 keys:
  type: string
  description: the SSH keys to be installed on the servers
  default: GiLAN
 dns_nameserver:
   type: string
   description: the IP address of YOUR local/global dns nameserver.
   default: "8.8.8.8"
 heat_net_mgmt:
   type: string
   description: mgmt network
   default: HEAT_NET_MNGT
 heat_net_an_lte:
   type: string
   description: an_lte network
   default: HEAT_AN_LTE

resources: #in this section, the resources are defined
# PORTS for MAC-addresses
# #-- Client_dave_2
 heat_client_dave_2_net_mgmt_port:
  type: OS::Neutron::Port
  properties:
    network_id: { get_param: heat_net_mgmt }
    mac_address: 52:54:00:82:d4:47
    port_security_enabled: false
    fixed_ips:
      - ip_address: 192.168.254.103

```

```
  heat_client_dave_2_net_an_lte_port:
   type: OS::Neutron::Port
   properties:
     network_id: { get_param: heat_net_an_lte }
     mac_address: 52:54:00:90:61:b5
     port_security_enabled: false
     fixed_ips:
       - ip_address: 192.168.3.103
 #SERVER
  heat_client_dave_2:
   type: OS::Nova::Server
   depends_on: [heat_client_dave_2_net_mgmt_port,
      heat_client_dave_2_net_an_lte_port ]
   properties:
    image: { get_param: image_client_dave_2 }
    name: INST_ROAMING_OP2_DAVE
    flavor: { get_param: flavor }
    key_name: { get_param: keys }
    networks:
     - port: {get_resource: heat_client_dave_2_net_mgmt_port}
     - port: {get_resource: heat_client_dave_2_net_an_lte_port}
```

CLIENT_EMMA

```
heat_template_version: 2014-10-16

description: BISDN GmbH, 2017. Requires existing public network
    named 'public' Template to deploy GiLAN clients. Should work
    with an operator already deploy.

parameters: #this section is if you want to give the parameters when
     you are creating the stack
 image_client_emma_2:
  type: string
  label: Image name or ID
  description: Image to be used for compute instance
  default: client-emma_2 #CirrOS 0.3.5 DT uses "CentOS 7", img
      available in our openstack
 flavor: #the flavor used
  type: string
  label: flavor
  description: flavor to be used
  default: openepc
 keys:
  type: string
  description: the SSH keys to be installed on the servers
  default: GiLAN
 dns_nameserver:
   type: string
   description: the IP address of YOUR local/global dns nameserver.
   default: "8.8.8.8"
 heat_net_mgmt:
   type: string
   description: mgmt network
   default: HEAT_NET_MNGT
 heat_net_an_lte:
   type: string
   description: an_lte network
   default: HEAT_AN_LTE

resources: #in this section, the resources are defined
# PORTS for MAC-addresses
# #-- Client_emma_2
 heat_client_emma_2_net_mgmt_port:
  type: OS::Neutron::Port
  properties:
    network_id: { get_param: heat_net_mgmt }
    mac_address: 52:54:00:45:2d:1c
    port_security_enabled: false
    fixed_ips:
      - ip_address: 192.168.254.104

```

```
  heat_client_emma_2_net_an_lte_port:
   type: OS::Neutron::Port
   properties:
     network_id: { get_param: heat_net_an_lte }
     mac_address: 52:54:00:9d:f5:e7
     port_security_enabled: false
     fixed_ips:
       - ip_address: 192.168.3.104
 #SERVER
  heat_client_emma_2:
   type: OS::Nova::Server
   depends_on: [heat_client_emma_2_net_mgmt_port,
      heat_client_emma_2_net_an_lte_port ]
   properties:
    image: { get_param: image_client_emma_2 }
    name: INST_ROAMING_OP2_EMMA
    flavor: { get_param: flavor }
    key_name: { get_param: keys }
    networks:
     - port: {get_resource: heat_client_emma_2_net_mgmt_port}
     - port: {get_resource: heat_client_emma_2_net_an_lte_port}
```